A Hybrid Metaheuristic approach to Search Based Stress Test

Francisco Nauber Bernardo Gois
Serviço Federal de Processamento de Dados
Avenida Pontes Vieira ,832, Fortaleza, Ceará, Brazil
francisco.gois@serpro.gov.br

Pedro Porfírio Muniz de Farias
Universidade de Fortaleza
Av. Washington Soares, 1321, Fortaleza, Ceará, Brazil
porfirio@unifor.br

André L. V. Coelho
Universidade de Fortaleza, Av. Washington Soares, 1321
Fortaleza, Ceará, Brazil
acoelho@unifor.br

Thiago Monteiro Barbosa
Serviço Federal de Processamento de Dados
Avenida Pontes Vieira ,832, Fortaleza, Ceará, Brazil
thiago.monteiro@serpro.gov.br

***Abstract*—Some software systems must respond to thousands or millions of concurrent requests. These systems must be properly tested to ensure that they can function correctly under the expected load. A common use of stress testing is to find test scenarios that produce execution times that violate the timing constraints specified. In this context, search-based testing is seen as a promising approach for verifying timing constraints. The main purpose of this paper is determine if hybrid algorithms are superior to single metaheuristics in search-based stress testing. The proposed hybrid metaheuristic approach uses genetic algorithms, simulated annealing, and tabu search algorithms in a stress testing model. The secondary objective of this paper is to improve stress testing automation. A tool named IAdapter, a JMeter plugin used for performing search-based stress tests, was developed. Two experiments were conducted to validate the proposed approach.**

I. Introduction

Many systems must support concurrent access by hundreds or thousands of users. Failure to providing scalable access to users may results in catastrophic failures and unfavorable media coverage [1].

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 [2].

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 [3] [1].

Stress testing determines the responsiveness, throughput, reliability, or scalability of a system under a given workload. The quality of the results of applying a given load testing to a system is closely linked to the implementation of the workload strategy. The performance of many applications depends on the load applied under different conditions. In some cases, performance degradation and failures arise only in stress conditions [4] [1].

978-1-5090-1633-4/16/$31.00

A stress test uses a set of workloads that consist of many types of usage scenarios and a combination of different numbers of users. A load is typically based on an operational profile. Different parts of an application should be tested under various parameters and stress conditions [5]. The correct application of a stress test should cover most parts of an application above the expected load conditions[3].

Fig. 1 shows an example of a system under assessment with three pages (the main page, profile page, and search page) and six possible users. From the combinations of users and application pages, various scenarios can be created, such as scenarios 1 and 2 shown in the figure. The first scenario presents a test that has passed, and the second scenario presents a test that has an HTTP 404 error.

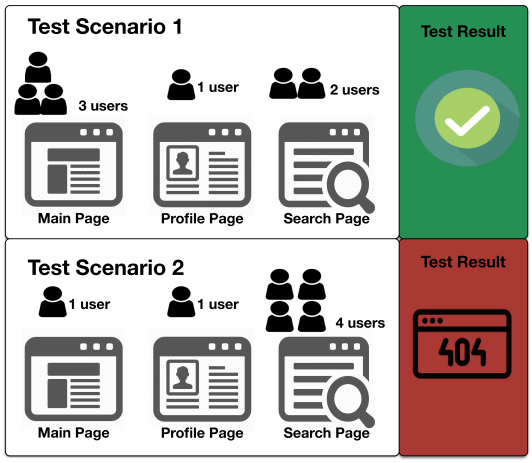

Figure 1: Possible test scenarios for a hypothetical application

A stress test usually lasts for several hours or even a few days and only tests a limited number of workloads. The major

challenge is to find the workloads that expose a major number of errors and to discover the maximum number of users supported by an application under testing [6].

Search-based testing is seen as a promising approach to verifying timing constraints [7]. A common objective of a load search-based test is to find scenarios that produce execution times that violate the specified timing constraints [8].

This paper has two main goals:

- To ascertain whether hybrid algorithms are superior to single metaheuristics when solving stress testing problem.
- To improve the process of stress testing with a tool and a test model that evolves during its execution.

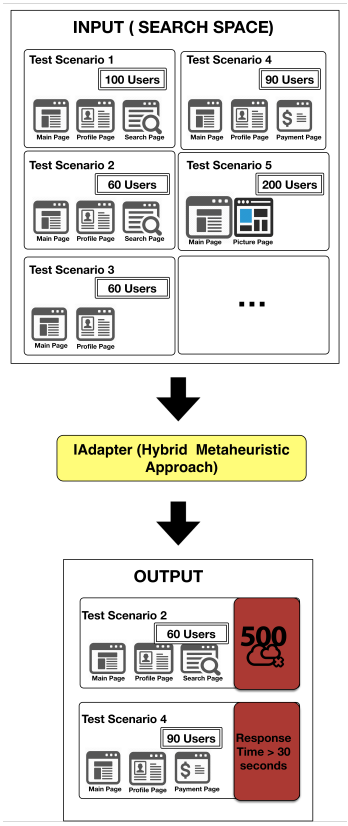

Figure 2: Illustrative example showing how IAdapter should be used

This paper proposes the use of a hybrid metaheuristic approach that combines genetic algorithms, simulated annealing, and tabu search algorithms in stress tests. A tool named IAdapter (www.iadapter.org, github.com/naubergois/newiadapter), a JMeter plugin for performing search-based load tests, was developed. Two experiments were conducted to validate the proposed approach. The first experiment was performed on an emulated component, and the second one was performed using an installed Moodle application.

Fig. 2 shows an example where IAdapter stress test automation finds two test scenarios. The first scenario presents a test that has an HTTP 500 error, and the second scenario presents a test that has a response time higher than 30 seconds.

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 concepts about search based tests. Section 5 presents concepts about metaheuristic algorithms.Section 6 presents concepts about hybrid metaheuristic algorithms. Section 7 discusses the related work. Section 8 presents the research-proposed approach. Section 9 presents the IAdapter tool. Section 10 shows the results of two experiments performed using the IAdapter plugin. Conclusions and further work are presented in Section 11.

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 [2].

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] [2].

The next subsections present details about the stress test process, automated stress test tools and the stress test results.

A. Stress Test Process

Contrary to functional testing, which has clear testing objectives, Stress testing objectives are not clear in the early development stages and are often defined later on a case-by-case basis. The Fig. 3 shows a commom Load, Performance and Stress test process [1].

The goal of the load design phase is to devise a load, which can uncover non-functional problems. Once the load is defined, the system under test executes the load and the system behavior under load is recorded. Load testing practitioners then analyze the system behavior to detect problems [1].

Once a proper load is designed, a load test is executed. The load test execution phase consists of the following three main aspects: (1) Setup, which includes system deployment and test execution setup; (2) Load Generation and Termination, which consists of generating the load; and (3) Test Monitoring and

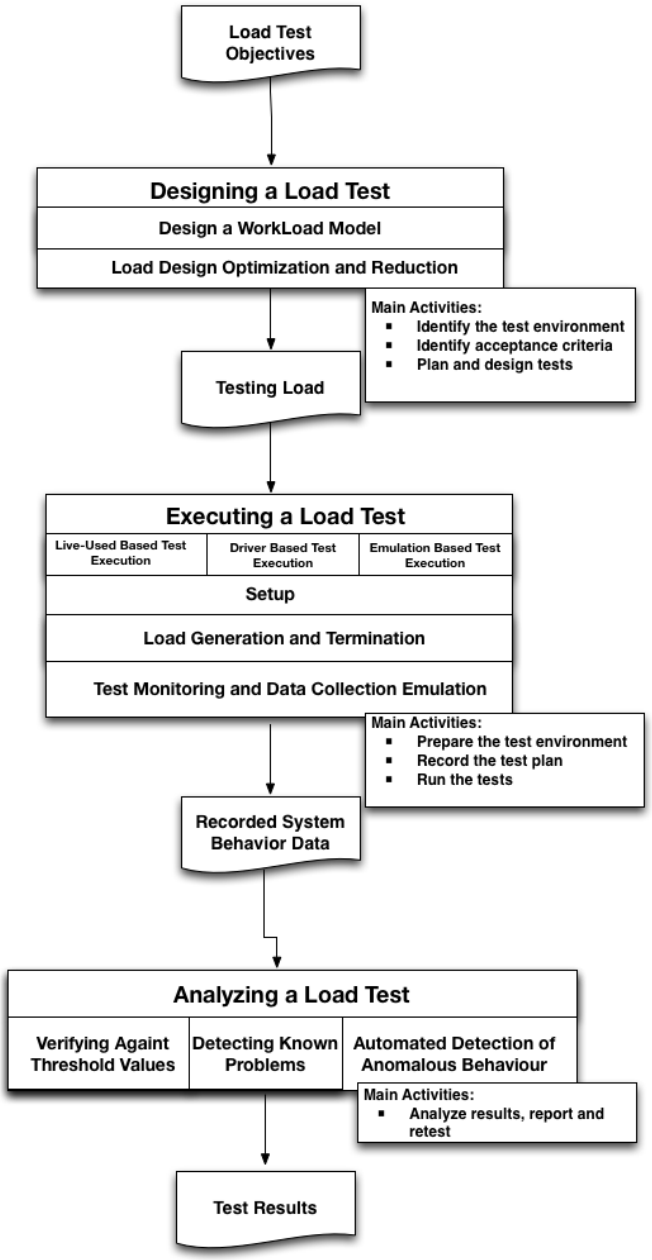

Figure 3: Load, Performance and Stress Test Process [1][12]

Data Collection, which includes recording the system behavior during execution[1].

The core activities in conducting an usual Load, Performance and Stress tests are [12]:

- Identify the test environment: identify test and production environments and knowing the hardware, software, and network configurations helps derive an effective test plan and identify testing challenges from the outset.
- Identify acceptance criteria: identify the response time, throughput, and resource utilization goals and constraints.
- Plan and design tests:identify the test scenarios.In the context of testing, a scenario is a sequence of steps in an application. It can represent a use case or a business function such as searching a product catalog, adding an item to a shopping cart, or placing an order [10].
- Prepare the test environment: configure the test environment, tools, and resources necessary to conduct the planned test scenarios.
- Record the test plan: record the planned test scenarios using a testing tool.
- Run the tests: Once recorded, execute the test plans under light load and verify the correctness of the test scripts and output results.
- Analyze results, report, and retest: examine the results of each successive run and identify areas of bottleneck that need addressing.

B. Automated Stress Test Tools

Automated tools are needed to carry out serious load, stress, and performance testing. Sometimes, there is simply no practical way to provide reliable, repeatable performance tests without using some form of automation. The aim of any automated test tool is to simplify the testing process. Automated Test Tool typically have the following components [2]:

- Scripting module: Enable recording of end-user activities in different middleware protocols;
- Test management module: Allows the creation of test scenarios;
- Load injectors: Generate the load with multiple workstations or servers;
- Analysis module: Provides the ability to analyse the data collected by each test interation.

Apache JMeter is a free open source stress testing tool. It has a large user base and offers lots of plugins to aid testing. JMeter is a desktop application designed to test and measure the performance and functional behavior of applications. The application it's purely Java-based and is highly extensible through a provided API (Application Programming Interface). JMeter works by acting as the client of a client/server application. JMeter allows multiple concurrent users to be simulated on the application [13] [12].

JMeter has components organized in a hierarchical manner. The Test Plan is the main component in a JMeter script. A typical test plan will consist of one or more Thread Groups, logic controllers, listeners, timers, assertions, and configuration elements:

- Thread Group: Test management module responsible to simulate the users used in a test. All elements of a test plan must be under a thread group.
- Listeners: Analysis module responsible to provide access to the information gathered by JMeter about the test cases .
- Samplers: Load injectors module responsible to send requests to a server, while Logical Controllers let you customize its logic.
- Timers: allow JMeter to delay between each request.

- Assertions: test if the application under test it is returning the correct results.
- Configuration Elements: configure detais about the request protocol and test elements.

C. Stress Test Results

The system behavior recorded during the test execution phase needs to be analyzed to determine if there are any load-related functional or non-functional problems [1].

There can be many formats of system behavior like resource usage data or end-to-end response time, which is recorded as response time for each individual request. These types of data need to be processed before comparing against threshold values.A proper data summarization technique is needed to describe these many data instances into one number. Some researchers advocate that the 90-percentile response time is a better measurement than the average/medium response time, as the former accounts for most of the peaks, while eliminating the outliers [1].

III. WorkLoad Model

Load, performance, or 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 [2].

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 [14] [2] [15].

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 [14] [2].

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 [14] [2].

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 (Fig. 4). This model is applied to all the workload attributes, e.g., computation, memory usage, I/O behavior, communication, etc. [11]. Fig. 4 shows a simplified workflow of a descriptive model. The workflow has six phases. In the first phase, the user uses the system in the production environment. In the second phase, the tester collects the user's data, such as logs, clicks, and preferences, from the system. The third phase consists in developing a model designed to emulate the user's behavior. The fourth phase is made up of the execution of the test, emulation of the user's behavior, and log gathering.

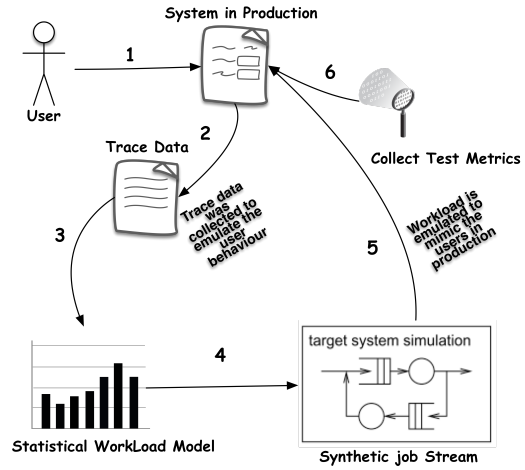

Figure 4: Workload modeling based on statistical data [11]

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 (Fig. 5).

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 [1]. The presented research work uses a generative model.

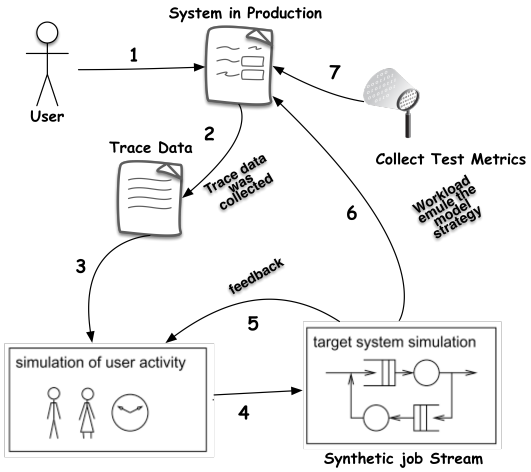

Figure 5: Workload modeling based on the generative model [11]

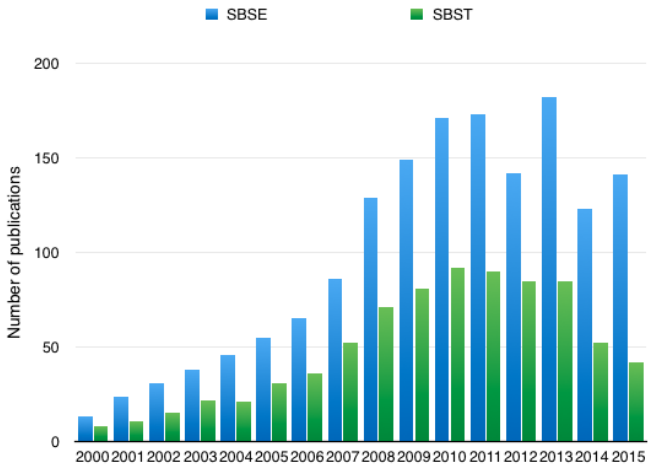

Figure 6: Number of publications in SBSE and SBST by Year. Data comes from the Harman et al., Afzal et al. and the SBSE repository [7] [16]

IV. Search Based Tests

Search-based software engineering (SBSE) is the application of optimization techniques in solving software engineering problems [1,2]. The applicability of optimization techniques in solving software engineering problems is suitable as these problems frequently encounter competing constraints and require near optimal solutions [7] [16].

Search Based Software Testing (SBST) is the sub-area of Search Based Software Engineering concerned with software testing. Search-based software testing is the application of metaheuristic search techniques to generate software tests. SBSE uses computational search techniques to tackle software engineering problems, typified by large complex search spaces. SBSE derives test inputs for a software system with the goal of improving various criteria. 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 [7] [17] [16].

Figure 6 shows the growth in papers published on SBST and SBSE. The data is taken from the SBSE repository (http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/) [130]. The aim of the SBSE repository is to contain every SBSE paper. Although no repository can guarantee 100% precision and recall, the SBSE repository has proved sufficiently usable that it has formed the basis of several other detailed analyses of the literature [16].

SBST has made many achievements, and demonstrated its wide applicability and increasing uptake. Nevertheless, there are pressing open problems and challenges that need more attention like to extend SBST to test non-functional properties, a topic that remains relatively under-explored, compared to structural testing. The Fig. 7 shows the non-funtional SBST by year [17] [16].

There are many kinds of non-functional search based tests [7]:

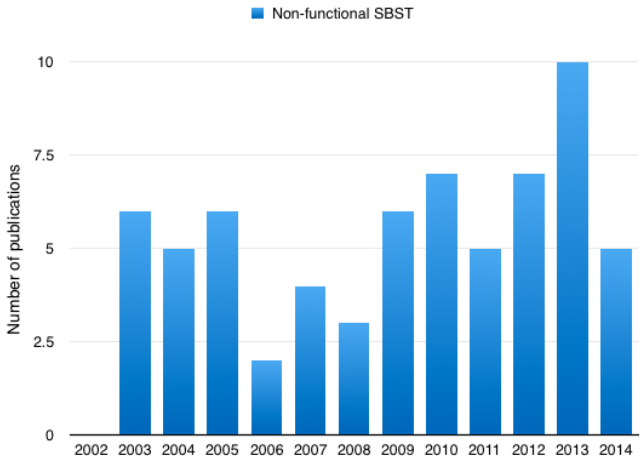

Figure 7: Number of publications in non-functional SBST by Year. Data comes from the Harman et al., Afzal et al. and the SBSE repository [7] [16]

- Execution time: The application of evolutionary algorithms to find the best and worst case execution times (BCET, WCET).
- Quality of service: uses metaheuristic search techniques to search violations of service level agreements (SLAs).
- Security: apply a variety of metaheuristic search techniques to detect security vulnerabilities like detecting buffer overflows.
- Usability: concerned with construction of covering array which is a combinatorial object.
- Safety: Safety testing is an important component of the testing strategy of safety critical systems where the systems are required to meet safety constraints.

A variety of metaheuristic search techniques are found to

be applicable for non-functional testing including simulated annealing, tabu search, genetic algorithms, ant colony methods, grammatical evolution, genetic programming and swarm intelligence methods.

V. Metaheuristics

In the computer science, the term metaheuristic is accepted for general techniques which are not specific to a particular problem. A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space [18].

Metaheuristics are strategies that guide the search process to efficiently explore the search space in order to find optimal solutions. Metaheuristic algorithms are approximate and usually non-deterministic and sometimes incorporate mechanisms to avoid getting trapped in confined areas of the search space. There are different ways to classify and describe metaheuristic algorithm [19]:

- Nature-inspired vs. non-nature inspired. There are nature-inspired algorithms, like Genetic Algorithms and Ant Algorithms, and non nature-inspired ones such as Tabu Search and Iterated Local Search.
- Population-based vs. single point search. Algorithms working on single solutions are called trajectory methods, like Tabu Search, Iterated Local Search and Variable Neighborhood Search. They all share the property of describing a trajectory in the search space during the search process. Population-based metaheuristics perform search processes which describe the evolution of a set of points in the search space.
- One vs. various neighborhood structures. Most metaheuristic algorithms work on one single neighborhood structure. In other words, the fitness landscape topology does not change in the course of the algorithm. Other metaheuristics, such as Variable Neighborhood Search (VNS), use a set of neighborhood structures which gives the possibility to diversify the search by swapping between different fitness landscapes.

Trajectory methods are characterized by a trajectory in the search space. Two common trajectory methods are Simulated Annealing and Tabu Search.

Simulated Annealing (SA) is a randomized algorithm that tries to avoid being trapped in local optimum solution by assigning probabilities to deteriorating moves. The SA procedure is inspired from the annealing process of solids. SA is based on a physical process in metallurgy discipline or solid matter physics. Annealing is the process of obtaining low energy states of a solid in heat treatment [20].

The algorithmic framework of SA is described in Alg. 1. The algorithm starts by generating an initial solution in function *GenerateInitialSolution()*. The initial temperature value is determined in function *SetInitialTemperature()* such that the probability for an uphill move is quite high at the start of the algorithm. At each iteration a solution s_1 is randomly chosen in function *PickNeighborAtRandom(N(s))*. If s_1 is better than s, then s_1 is accepted as new current solution. Else, if the move from s to s_1 is an uphill move, s_1 is accepted with a probability which is a function of a temperature parameter Tk and s [18].

Algorithm 1 Simulated Annealing Algorithm

```
1:  s ← GenerateInitialSolution()
2:  k ← 0
3:  Tk ← SetInitialTemperature()
4:  while termination conditions not met do
5:      s1 ← PickNeighborAtRandom(N(s))
6:      if (f(s1) < f(s)) then
7:          s ← s1
8:      else Accept s1 as new solution with probability
    p(s1|Tk,s)
9:      end if
10:     K ← K + 1
11:     Tk ← AdaptTemperature()
12: end while
```

Algorithm 2 Tabu Search Algorithm

```
    s ← GenerateInitialSolution()
2:  InitializeTabuLists(TL1,...,TLr)
    while termination conditions not met do
4:      Na(s) ← {s1 ∈ N(s)|s1 does not violate a tabu condition, or it satisfies at least one aspiration condition }
        s1 ← argmin{f(s2)|s2 ∈ Na(s)}
6:      UpdateTabuLists(TL1,...,TLr,s,s1)
        s ← s1
8:  end while
```

Tabu Search is a metaheuristic that guides a local heuristic search procedure to explore the solution space beyond local optimal and search with short term memory to avoid cycles. Tabu Search uses a tabu list to keep track of the last moves, and dont allow going back to these [21].

The algorithmic framework of Tabu Search is described in Alg. 2. The algorithm starts by generating an initial solution in function *GenerateInitialSolution()* and the tabu lists are initialized as empty lists in function *InitializeTabuLists(TL_1,...,TL_r)*. For performing a move, the algorithm first determines those solutions from the neighborhood *N(s)* of the current solution s that contain solution features currently to be found in the tabu lists. They are excluded from the neighborhood, resulting in a restricted set of neighbors $N_a(s)$. At each iteration the best solution s_1 from $N_a(s)$ is chosen as the new current solution. Furthermore, in procedure *UpdateTabuLists(TL_1,...,TL_r,s,s_1)* the corresponding features of this solution are added to the tabu lists.

Population-based metaheuristics (P-metaheuristics) could be viewed as an iterative improvement in a population of solutions. First, the population is initialized. Then, a new population of solutions is generated. Finally, this new population is integrated into the current one using some selection procedures. The search process is stopped when a stopping criterion is satisfied. Algorithms such as Genetic algorithms

Algorithm 3 Genetic Algorithm

$s \leftarrow GenerateInitialSolution()$
Evaluate(P)
while termination conditions not met **do**
  $P_1 \leftarrow Recombine(P)$
  $P_2 \leftarrow Mutate(P_1)$
  $Evaluate(P_2)$
  $P \leftarrow Select(P_2, P)$
end while

(GA), scatter search (SS), estimation of distribution algorithms (EDAs), particle swarm optimization (PSO), bee colony (BC), and artificial immune systems (AISs) belong to this class of metaheuristics [22].

Algorithm 3 shows the basic structure of GA algorithms. In this algorithm, P denotes the population of individuals. A population of offspring is generated by the application of recombination and mutation operators and the individuals for the next population are selected from the union of the old population and the offspring population [18].

VI. Hybrid Metaheuristics

A combination of one metaheuristic with components from other metaheuristics is called a hybrid metaheuristic. The concept of hybrid metaheuristics has been commonly accepted only in recent years, even if the idea of combining different metaheuristic strategies and algorithms dates back to the 1980s. Today, we can observe a generalized common agreement on the advantage of combining components from different search techniques and the tendency of designing hybrid techniques is widespread in the fields of operations research and artificial intelligence [18].

There are two main categories of metaheuristic combinations: collaborative combinations and integrative combinations. These are presented in Fig. 8 [23].

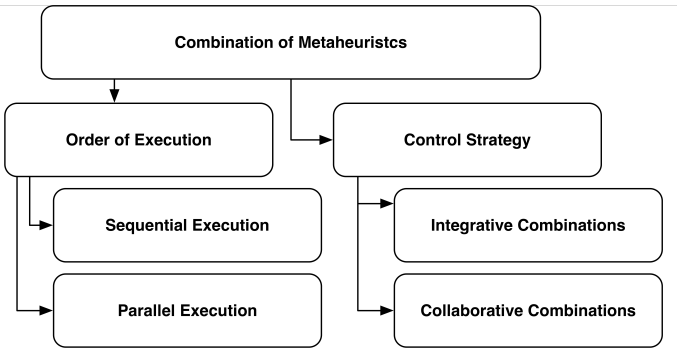

Figure 8: Categories of metaheuristc combinations [24]

Collaborative combinations use an approach where the algorithms exchange information, but are not part of each other. In this approach, algorithms may be executed sequentially or in parallel.

One of the most popular ways of metaheuristic hybridization consists in the use of trajectory methods inside population-based methods. Population-based methods are better in identifying promising areas in the search space from which trajectory methods can quickly reach good local optima. Therefore, metaheuristic hybrids that can effectively combine the strengths of both population-based methods and trajectory methods are often very successful [18].

The work uses a type of collaborative combination with sequential execution with two trajectory methods (Tabu Search and Simulated Annealing) and Genetic Algorithms.

VII. Related Work

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 [8].

A common goal of search-based stress testing is to find test scenarios that produce execution times that exceed the timing constraints specified. If a temporal error is found, the test was successful [8]. The application of evolutionary algorithms to stress tests involves finding the best- and worst-case execution times (B/WCET) to determine whether timing constraints are fulfilled [7].

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 [7] [25].

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 [7]. However, stress testing where testers have no access to the production environment should be measured by the execution time measurement [2] [7].

Table I shows a comparison between the presented research work and the research studies on load, performance, and stress tests presented by Afzal et al. [26]. Afzal's work adds to some of the latest research in this area ([27] [4] [28] [29] [30]).

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). Most research studies are limited to making prototypes of genetic algorithms. The presented research work is distinguished from others by having a functional tool using a hybrid approach.

Wegener et al. [32] 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 [32].

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

	Prototypes		Functional Tool
	Execution Time	Processor Cycles	Execution Time
GA + SA + Tabu Search			IADAPTER
GA	Alander et al.,1998 [31] Wegener et al., 1996 a 1997 [32][33] Sullivan et al., 1998 [8] Briand et al., 2005 [34] Canfora et al., 2005 [35]	Wegener and Grochtman 1998 [36] Mueller et al., 1998 [37] Puschner et al. [38] Wegener et al., 2000 [39] Gro et al., 2000 [40]	Di Penta, 2007 [41] Garoussi, 2006 [27] Garousi, 2008 [42] Garousi, 2010 [4]
Simulated Annealing (SA)			Tracey, 1998 [43]
Constraint Programming			Alesio, 2014 [29] Alesio, 2013 [28]
GA + Constraint Programming			Alesio, 2015 [30]
Customized Algorithm		Pohlheim, 1999 [44]	

Alander et al. [31] 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 [31].

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 [33] [36] .

Gro et. al. [40] 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 [40].

Tracey et al. [43] 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 [43].

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[44].

Briand et al. [34] 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 [34].

Di Penta et al. [41] 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, applied the black-box approach with fitness calculated only on the basis of how close solutions violate QoS constraint. In case of audio workflow, the GA outperformed random search. For the second case study, use of black-box approach successfully violated the response time constraint, showing the violation of QoS constraints for a real service available on the Internet [41].

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 [27].

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 [30].

The presented research work and Alesio's approach [30] use a hybrid approach with a functional tool. Table II presents the main differences between Alesio's and IAdapter's approaches. Whereas the present research uses an approach based on usage scenarios performing tests on an application installed in an available environment, Alesio uses sequence diagrams to select for arrival time of tasks in systems from safety-critical domains.

Table II: Main differences between Alesio's [30] and IAdapter's approaches

	Alesio et al. [30]	IAdapter
Metaheuristcs	GA+ Constraint Programming	GA+SA+ Tabu Search
Inputs	Design Model (Time and Concurrency Information)	Number of Users Ramp-up Test scenarios
Main Objective	Find task arrival times of aperiodic tasks that maximizing deadline misses	Find the number of users, ramp-up and test scenarios that maximizing deadline misses
Main Application	Systems from safety-critical domains	Web and Mobile applications

VIII. Improving Stress Search-Based Testing Using a Hybrid Metaheuristic Approach

A large number of researchers have recognized the advantages and huge potential of building hybrid metaheuristics. The main motivation for creating hybrid metaheuristics is to exploit the complementary character of different optimization strategies. In fact, choosing an adequate combination of algorithms can be the key to achieving top performance in solving many hard optimization problems [24] [45].

The proposed solution makes it possible to create a model that evolves during the test. The proposed solution model uses genetic algorithms, tabu search, and simulated annealing in two different approaches. The study initially investigated the use of these three algorithms. Subsequently, the study will focus in others Population-based and single point search metaheuristics. The first approach uses the three algorithms independently, and the second approach uses the three algorithms collaboratively (hybrid metaheuristic approach).

In the first approach , the algorithms do not share their best individuals among themselves. Each algorithm evolves in a separate way (Fig. 9). The second approach uses the algorithms in a collaborative mode (hybrid metaheuristic). In this approach, the three algorithms share their best individuals found (Fig. 10).

The next subsections present details about the used metaheuristic algorithms (Representation, initial population and fitness function).

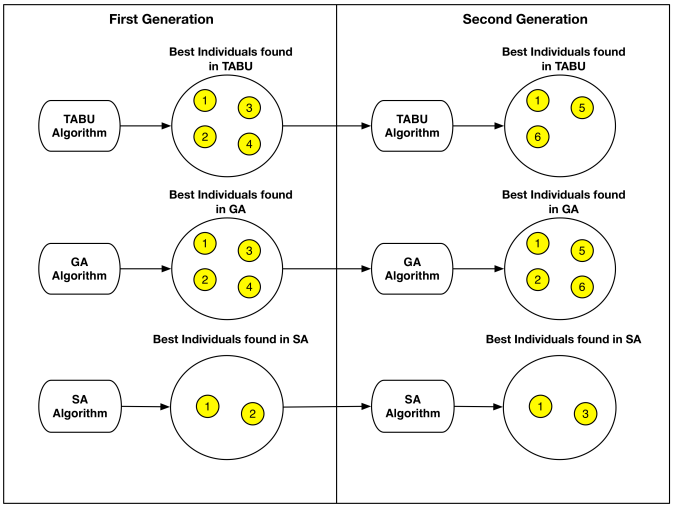

Figure 9: Use of the algorithms independently

A. Representation

The solution representation is composed by a linear vector with 23 positions. The first position represents the name of an individual. The second position represents the algorithm (genetic algorithm, simulated annealing, or Tabu search) used by the individual. The third position represents the type of test (load, stress, or performance). The next positions represent 10 scenarios and their numbers of users. Each scenario is an atomic operation: the scenario must log into the application, run the task goal, and undo any changes performed, returning the application to its original state.

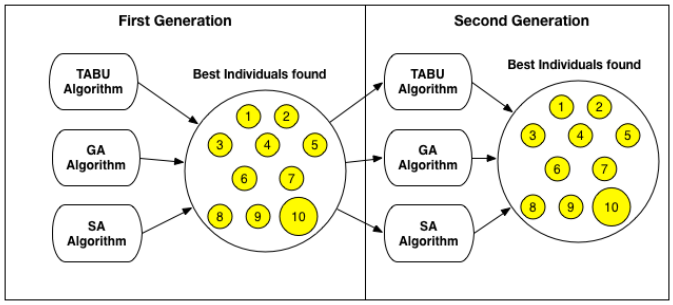

Figure 10: Use of the algorithms collaboratively

Fig. 11 presents the solution representation and an example using the crossover operation. In the example, genotype 1 has the Login scenario with 2 users, the Form scenario with 0 users, and the Search scenario with 3 users. Genotype 2 has the Delete scenario with 10 users, the Search scenario with 0 users, and the Include scenario with 5 users. After the crossover operation, we obtain a genotype with the Login scenario with 2 users, the Search scenario with 0 users, and the Include scenario with 5 users.

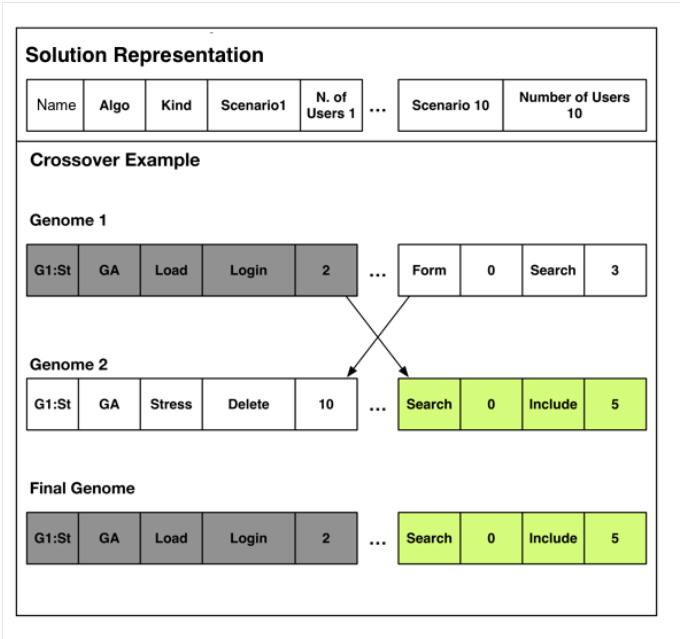

Figure 11: Solution representation and crossover example

Fig. 12 shows the strategy used by the proposed solution to obtain the representation of 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.

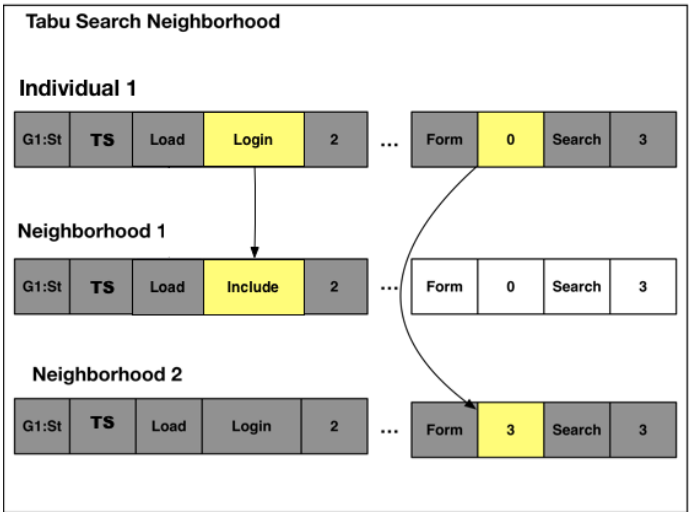

Figure 12: Tabu search and simulated annealing neighbor strategy

B. Initial population

The strategy used by the plugin to instantiate the initial population is to generate 50% of the individuals randomly, and 50% of the initial population is distributed in three ranges of values:

- Thirty percent of the maximum allowed users in the test;
- Sixty percent of the maximum allowed users in the test; and
- Ninety percent of the maximum allowed users in the test.

The percentages relates to the distribution of the users in the initial test scenarios of the solution. For example, in a hypothetical test with 100 users, the solution will create initial test scenarios with 30, 60 and 90 users.

C. Objective (fitness) function

The proposed solution was designed to be used with independent testing teams in various situations where the teams have no direct access to the environment where the application under test was installed. Therefore, the IAdapter plugin uses a measurement approach to the definition of the fitness function. The fitness function applied to the IAdapter solution is governed by the following equation:

$$\begin{aligned} fit = 90percentileweigth * 90percentiletime \\ +80percentileweigth * 80percentiletime \\ +70percentileweigth * 70percentiletime + \\ maxResponseWeigth * maxResponseTime + \\ numberOfUsersWeigth * numberOfUsers - penalty \end{aligned} \quad (1)$$

The proposed solution's fitness function uses a series of manually adjustable user-defined weights (90percentileweight, 80percentileweight, 70percentileweight, maxResponseWeight, and numberOfUsersWeight). These weights make it possible to customize the search plugin's functionality. A penalty is applied when an application under test takes a longer time to respond than the level of service.

IX. IAdapter

IAdapter is a JMeter plugin designed to perform search-based stress tests. The plugin is available on www.iadapter.org. The IAdapter plugin implements the solution proposed in Section 5. The next subsections present details about the Apache JMeter tool, the IAdapter Life Cycle and the IAdapter Components. The IAdapter plugin provides three main components: WorkLoadThreadGroup, WorkLoadSaver, and WorkLoadController.

The Fig. 13 show the IAdapter architecture. All metaheuristic class implements the interface IAlgorithm. Test scenarios and test results are stored in a Mysql database. GeneticAlgorithm class uses a framework named JGAP to implement Genetic Algorithms.

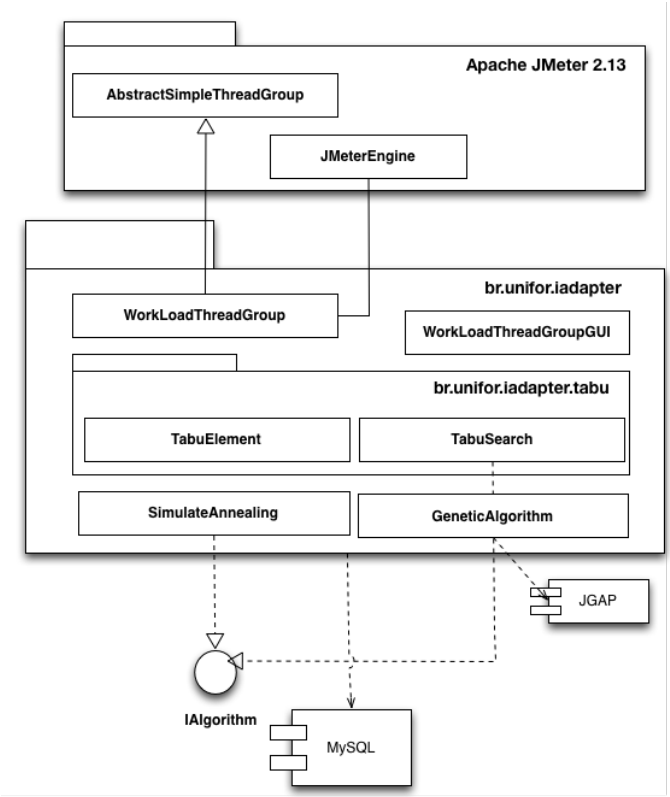

Figure 13: IAdapter architecture

The WorkLoadThreadGroup class is the Load Injection and Test Management modules, responsible to generate the initial population and uses the JMeter Engine to realize requests to server under test.

A. IAdapter Life Cycle

Fig. 14 presents the IAdapter Life Cycle. The main difference between IAdapter and JMeter tool is that the IAdapter provide an automated test execution where the new test scenarios are choosen by the test tool. In a test with JMeter, the tests scenarios are usually chosen by a test designer.

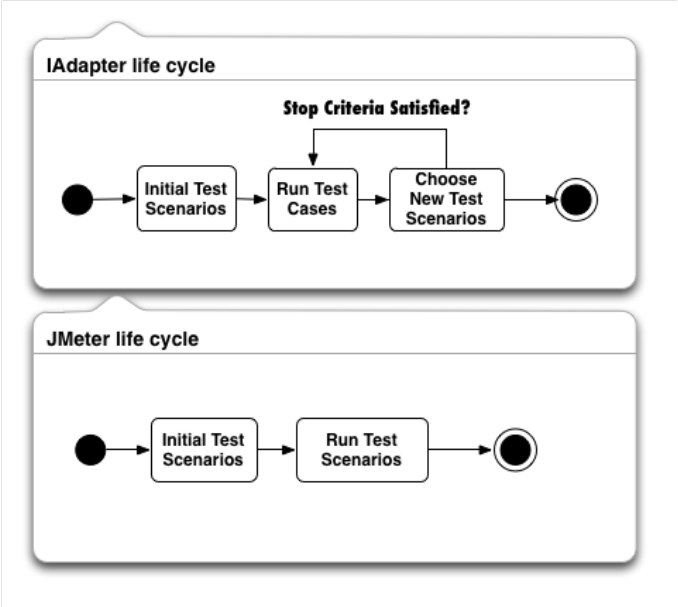

Figure 14: IAdapter life cycle

Listing 1: TabuSearch class

```
1  public class TabuSearch {
2
3    private static int tabuExpires;
4
5    private static List<TabuElement> tabuTable = new
         ArrayList<TabuElement>();
6
7    public static List<WorkLoad> verify(List<WorkLoad>
         list, List<TestElement> nodes) {
8      //ArrayList with the workloads to remove
9      List<WorkLoad> workLoadForRemove = new ArrayList<
           WorkLoad>();
10     for (WorkLoad workLoad : list) {
11       //Add elements in the tabu list to remotion
12       for (TabuElement tabuElement : tabuTable) {
13         TabuElement tabu = WorkLoadUtil.convertTabu(
               workLoad, nodes);
14         workLoadForRemove.add(workLoad);      }
15     }
16     TabuSearch.setTabuExpires(TabuSearch.getTabuExpires
           () + 1);
17 !\colorbox{light-gray}{new MatrixInt}!    if (
       TabuSearch.getTabuExpires() > expiresCriteria)
18     {
19       TabuSearch.setTabuTable(new ArrayList<TabuElement
             >());
20       TabuSearch.setTabuExpires(0);
21     }
22     list.removeAll(workLoadForRemove);
23     return list;
24   }
```

B. IAdapter Components

WorkLoadThreadGroup is a component that creates an initial population and configures the algorithms used in IAdapter. Fig. 15 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.

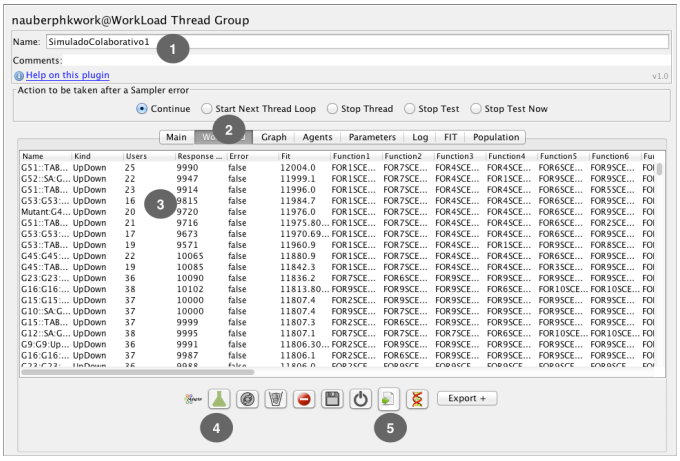

Figure 15: WorkLoadThreadGroup component

WorkLoadThreadGroup component uses the GeneticAlgorithm, TabuSearch and SimulateAnnealing classes. The Listing 1 shows the Tabu Search class. The class has one main method named verify that remove all workload that are contained in the tabu list. The workloads are removed from tabu list when a expiration criteria is attended.

The Listing 2 shows the Simulated Annealing implementation. The algorithm iterate over a set of new places and accept new solutions with a better fitness value. Each new place represents a new workload in the stress test.

The WorkLoadSaver component is responsible for saving all data in the 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.

X. Experiments

This section presents two experiments. The first one was performed on an emulated component, and the second one was performed using an installed Moodle application. The experiments used the following fitness function:

$$\begin{aligned} fit = {} & 0.9 * 90 percentiletime \\ & +0.1 * 80 percentiletime \\ & +0.1 * 70 percentiletime + \\ & 0.1 * maxResponseTime + \\ & 0.2 * numberOfUsers - penalty \end{aligned} \quad (2)$$

This fitness function is the same function represented in the section VII with the manually adjustable user-defined weights filled out. This fitness function intended to find individuals with the highest percentile of 90%, followed by individuals with a higher percentile time of 80% and 70%, maximum response time, and number of users.

Listing 2: Simulated Annealing Implementation

```
1  //Simulated Annealing Method
2  public static int sa(int users, List<WorkLoad>
       newPlaces, int maxUsers,
3        List<WorkLoad> list, int generation,
           WorkLoadThreadGroup tg,
4        List<TestElement> nodes) {
5      //Set Initial Temperature
6      int newUsers = users;
7      //Iterate in new places
8      for (WorkLoad newPlace : newPlaces) {
9
10       WorkLoad place = tg.getWorkloadCurrentSA();
11       if (users > 0) {
12         if ((place != null) && (newPlace != null)) {
13           double deltaC = place.getFit() - newPlace.
               getFit();
14           //If the new place has a better fitnesse
               value
15           //Accept new solution
16           if (deltaC < 0) {
17             tg.setWorkloadCurrentSA(newPlace);
18
19           } else {
20
21             Random random = new Random();
22             double randomDouble = random.nextDouble();
23             SimulateAnnealing.tries += 1;
24             double exponential = Math.exp(-1 * (deltaC
                 / users));
25             //Accept new solution with probability
26             if (randomDouble > exponential) {
27               tg.setWorkloadCurrentSA(newPlace);
28
29             }
30 ...
```

Listing 3: SimulateConcurrentAccess class

```
1  public class SimulateConcurrentAccess {
2    @Test
3    public void firstScenario() {
4      synchronized (StaticClass.class) {
5        for (int i = 0; i <= 1000; i++) {
6          StaticClass.x += i;
7        }
8        StaticClass.x = 0;
9      }
10   }
11
12     @Test
13   public void secondScenario() {
14     synchronized (StaticClass.class) {
15       for (int i = 0; i <= 2000; i++) {
16         StaticClass.x += i;
17       }
18       StaticClass.x = 0;
19     }
20   }
```

The first experiment ran for 27 generations, and the second experiment performed 6 generations, with 300 executions by generation (100 times for each algorithm), generating 300 new individuals. The experiments used an initial population of 100 individuals. The genetic algorithm used the top 10 individuals from each generation in the crossover operation. The Tabu list was configured with the size of 10 individuals and expired every 2 generations. The mutation operation was applied to 10% of the population on each generation.

A. First Experiment: Emulated Class Test

The first experiment aimed to perform performance, load, and stress testing on a simulated component. The purpose of using a simulated component was to be able to perform a greater number of generations in a shorter time available and eliminate variables such as the use of databases and application servers. The first experiment used a test class named SimulateConcurrentAccess. This class has a static variable named x and a set of methods that use the variable in a synchronized context (Listing 3). The experiment was executed using the JMeter Java Request Sampler Component with IAdapter.

Fig.16 presents the best results in 27 generations applied in the first experiment. The figure shows the results obtained with the algorithms with and without collaboration. The x axis represents the generation number, and the y axis represents the best fitness value obtained until the current generation. A higher value in the figure means that the scenario has a greater response time by the application under test. The results of the experiment showed that the use of cooperation between the three algorithms resulted in finding the individuals with better fitness values.

Figure 16: Best results obtained in 27 generations

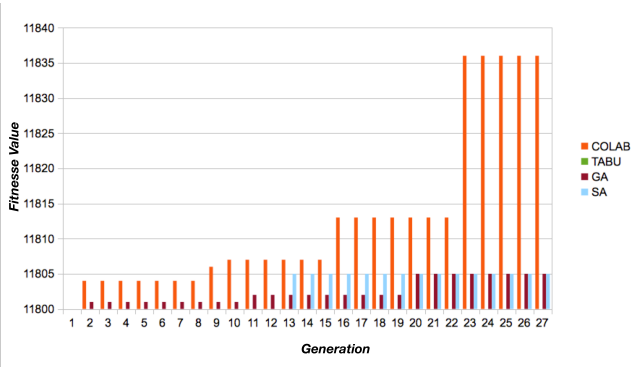

Table III presents the results obtained by the hybrid meta-heuristic (HM) approach, genetic algorithm (GA), simulated annealing (SA), and Tabu search (TS) from 27 generations in the first experiment. The values are the maximum fitness value obtained by each algorithm.

The signed-rank Wilcoxon non-parametrical procedure was used for comparing the results with Z-value and W-value. The significant level adopted was 0.05. The Z-value obtained was -2.2736 and the p-value was 0.0232. The W-value obtained was 78. The critical value of W for N = 25 at p 0.05 was 89.The result was significant at p 0.05. The procedure showed that there was a significant improvement in the results with the collaborative approach.

B. Second Experiment: Moodle Application Test

The second experiment used a Moodle application installed in a machine with 500 GB of hard disk space and 8 GB of memory. The study used six application scenarios:

- PostDeleteMessage: This scenario posts and deletes messages in the Moodle application.

Table III: Maximum value of the fitness function by algorithm

GEN	HM	TS	GA	SA
1	11238	11238	11238	11238
2	11804	11596	11801	10677
3	11787	8932	8411	10869
4	11723	9753	9611	10760
5	8164	9780	10738	4794
6	11802	9781	11086	6120
7	9985	5782	11272	11798
8	11803	11749	10084	11309
9	11806	7284	11633	10766
10	11807	9386	11717	4557
11	11802	9653	11802	11151
12	11807	10594	11793	9434
13	11802	10848	10382	11805
14	11801	11551	7219	10237
15	11807	1701	7189	9338
16	11813	6203	11758	5321
17	11805	10720	10805	11748
18	9600	6371	11698	7818
19	11733	8160	11648	11509
20	9589	9428	11805	4813
21	11800	9463	11798	10801
22	11805	11799	11804	6029
23	11836	11655	11800	3579
24	11805	11512	11803	5761
25	11804	11573	11802	9680
26	11800	11575	11403	9388
27	11805	10691	11745	9465

- MyHome: This scenario accesses the homepage of the user's application.
- Login: This scenario is responsible for user authentication by the application.
- Notifications: This scenario involves entering the notification page of each user.
- Start Page: This scenario shows the initial start page of the application.
- Badge: This scenario involves entering the badge page.

The maximum tolerated response time in the test was 30 seconds. Any individuals who obtained a time longer than the stipulated maximum time suffered penalties. 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.

Table IV presents the maximum fitness value obtained by the hybrid metaheuristic (HM) approach, genetic algorithm (GA), simulated annealing (SA), and Tabu search (TS) in each generation.

Table IV: Results obtained from the second experiment

GEN	HM	TS	GA	SA
1	32242	32242	32242	32242
2	34599	32443	26290	35635
3	35800	34896	34584	34248
4	35782	34912	32689	25753
5	35611	31833	34631	8366
6	35362	35041	33397	9706

The small number of samples of the experiment is insufficient to give a statistical significance to the results of the Wilcoxon procedure. However, it is noted that, in four of six generations, the collaborative approach presented the best values. The experiment succeeded in finding 29 individuals whose maximum time expected by the application was obtained. Table V shows an example of the six individuals with the highest fitness values in the second experiment. The table shows the fitness value (Fit); the name of the scenario (Scenario); the number of users (Users); and the percentiles of 90%, 80%, and 70% (90per, 80per and 70per) in seconds.

Table V: Example of individuals obtained in the second experiment

Id	Fit	Scenario	Users	90per	80per	70per
1	35800	MyHome	31	30	29	10
		Badges	4			
2	35795	MyHome	30	30	29	10
		Notifications	2			
		Badges	2			
3	35782	MyHome	32	30	29	10
		Badges	3			
4	35773	MyHome	22	30	29	10
		Notifications	6			
		Badges	9			
5	35771	MyHome	28	30	29	9
		Badges	6			
6	35683	MyHome	27	30	29	8
		Badges	10			

Table VI presents the percentage of genes in all test scenarios by generation with and without collaboration. Most of the genes converged to the MyHome feature, which had the highest application response time.

Table VI: Percentage of genes in each scenario by generation

Gen/ Scenarios	**Non collaboration approach**						
	Initial	1	2	3	4	5	6
Badges	20	18	16	24	15	16	17
MyHome	**15**	**59**	**55**	**48**	**53**	**50**	**52**
StartPage	15	10	12	11	20	18	19
Notifications	25	5	11	10	9	10	9
Post	8	3	1	3	1	2	1
Login	17	5	5	4	2	4	2
	Collaboration approach						
Badges	20	29	16	25	9	16	9
MyHome	**15**	**29**	**69**	**49**	**74**	**66**	**76**
StartPage	15	22	10	21	10	10	8
Nofications	25	10	1	1	2	1	3
Post	8	2	1	1	1	2	1
Login	17	8	3	3	4	5	3

XI. Conclusion

This paper presented a hybrid metaheuristic approach for use in stress testing. Two experiments were performed to validate the solution. The first experiment was performed on an emulated component, and the second experiment was performed using an installed Moodle application. The collaborative approach obtained better fit values in both experiments.

The main contributions of this research are as follows: The presentation of a hybrid metaheuristic approach for use in

stress tests; the development of a JMeter plugin for search-based tests; and the automation of the stress test execution process.

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.

There is a range of future improvements in the proposed approach. Also as a typical search strategy, it is difficult to ensure that the execution times generated in the experiments represents global optimum. More experimentation is also required to determine the most appropriate and robust parameters. Lastly, there is a need for an adequate termination criterion to stop the search process.

Among the future works of the research, the use of new combinatorial optimization algorithms such as very large-scale neighborhood search is one that we can highlight

References

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

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

[3] 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.

[4] V. Garousi, "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.

[5] C. Babbar, N. Bajpai, and D. Sarmah, "Web Application Performance Analysis based on Component Load Testing," *International Journal of Technology*, 2011.

[6] C. Barna, M. Litoiu, and H. Ghanbari, "Autonomic load-testing framework," *International conference on Autonomi*, pp. 91–100, 2011.

[7] 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.

[8] 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.

[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] B. Erinle, *Performance Testing With JMeter 2.9*, 2013.

[13] E. H. Halili, *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites.*, 2008.

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

[15] 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.

[16] M. Harman, Y. Jia, and Y. Zhang, "Achievements , open problems and challenges for search based software testing," *8th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, no. Icst, 2015. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/mharman/icst15.pdf

[17] A. Aleti, I. Moser, and L. Grunske, "Analysing the fitness landscape of search-based software testing problems," *Automated Software Engineering*, pp. 1–19, 2016.

[18] G. R. Raidl, J. Puchinger, and C. Blum, "Metaheuristic hybrids," in *Handbook of metaheuristics*. Springer, 2010, pp. 469–496.

[19] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 189–213, 2003.

[20] W. Jaziri, *Local Search Techniques: Focus on Tabu Search*, 2008.

[21] F. Glover and R. Martí, "Tabu Search," *Tabu Search*, pp. 1–16, 1986.

[22] E.-G. Talbi, *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009, vol. 74.

[23] R. Raidl, "A Unified View on Hybrid Metaheuristics," *Hybrid Metaheuristics (LNCS 4030)*, pp. 1–12, 2006.

[24] J. Puchinger and R. Raidl, "Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization : A Survey and Classification," *Artificial Intelligence and Knowledge Engineering Applications a Bioinspired Approach*, vol. 3562, pp. 41–53, 2005.

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

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

[27] V. Garousi, "Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models using Genetic Algorithms," no. August, 2006.

[28] 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.

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

[30] 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.

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

[32] 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

[33] 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.

[34] 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.

[35] 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."

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.
[42] 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.
[43] N. J. Tracey, J. a. Clark, and K. C. Mander, "Automated Programme Flaw Finding using Simulated Annealing," 1998.
[44] 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.
[45] C. Blum, "Hybrid metaheuristics in combinatorial optimization: A tutorial," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7505 LNCS, no. 6, pp. 1–10, 2012.