Improving Load, Performance and Stress Search Based Testing using a Hybrid Metaheuristic Approach

Francisco Nauber Bernardo Gois
Serviço Federal de Processamento de Dados
Avenida Pontes Vieria ,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 Luís Vasconcelos 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 Vieria ,832
Fortaleza, Ceará, Brazil
thiago.barbosa@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. Load, performance, and stress search-based testing aim to find test scenarios that produce execution times that violate the timing constraints specified. The purpose of this paper is to identify test scenarios that exercise a system in a way that tasks are pushed as close as possible to their limits. The research proposes a hybrid metaheuristic approach that uses genetic algorithms, simulated annealing, and Tabu search algorithms for use in load, performance, and stress test models. A tool named IAdapter, a JMeter plugin used for performing search-based load, performance, or stress 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. In both experiments, the use of a hybrid metaheuristic approach produced better fitness values.

CCS Concepts

•Software and its engineering → Software verification and validation;

Keywords

Search-Based Testing;Tabu Search;Hybrid Metaheuristics

1. INTRODUCTION

SBST '06 Austin, Texas USA

DOI: 10.475/123_4

Many systems must support concurrent access by hundreds or thousands of users. Failure to scale users results in catastrophic failures and unfavorable media coverage [23]. To ensure the quality of these systems, it is necessary to perform a testing procedure such as performance, stress, and load testing [24].

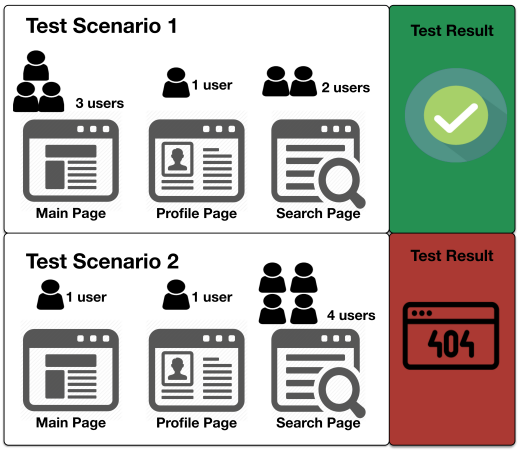
The explosive growth of the Internet has contributed to the increased need for applications that perform at an appropriate speed. Performance problems are often detected late in the application life cycle, and the later they are discovered, the greater the cost to fix them [26]. The use of load testing 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, generated by system users, can result in highly critical failures and negatively affect the customers's perception of the company [14] [23].

Load testing determines the responsiveness, throughput, reliability, or scalability of a system under a given workload. The quality of the results of a system's load tests 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 [19] [23].

A load, performance, or 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 on various parameters and stress conditions [5]. The correct application of a load test should cover most parts of an application under ordinary conditions (load or performance test) or above the expected load conditions (stress test)[14] [25] [15].

Fig. 1 shows an example of a system under test 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 error of 404.

Figure 1: Possible test scenarios for a hypothetical application



A performance 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 test [6].

Search-based test is seen as a promising approach for verifying timing constraints [2]. The main objective of a load, performance, and stress search-based test is to find test scenarios that produce execution times that violate the specified timing constraints [35].

The main objective of this research is to find test scenarios that maximize the likelihood of task deadline misses. Fig. 2 presents an illustrative example where the presented research work approach finds test scenarios that expose errors or response times exceeding the maximum response time expected. There are many benefits in the use of the solution:

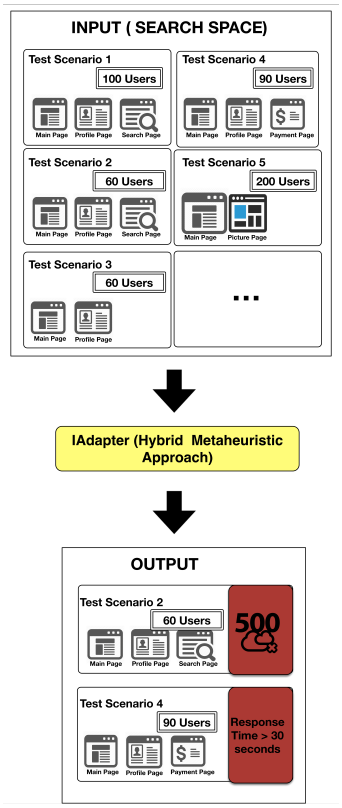
- It generates stress test cases that maximize the likelihood of task deadline misses.
- It automates the search for failure points of an application under test.

The paper proposes the use of a hybrid metaheuristic approach that combines genetic algorithms, simulated annealing, and Tabu search algorithms in load, performance, and stress evolutionary tests.

A tool named IAdapter, a JMeter plugin for performing search-based load, performance, or stress 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.

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

Figure 2: Illustrative example of the use of the presented research work approach



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 [34] [10].

The performance test 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 [13]. The purpose of this test is to demonstrate that the system reaches its performance objectives [34].

In a load test, the system is evaluated at predefined load levels [13]. The aim of this test is to determine whether the system can reach its performance targets for availability, concurrency, throughput, and response time. Load test is the closest to real application use [26].

The stress test verifies the system behavior against heavy workloads [34], 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 [13] [26].

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 [26].

2. LOAD, PERFORMANCE AND STRESS TEST

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].

Load, performance, and stress results are measured by indicators. 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 [23].

3. 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 [26].

The term ´workload´ represents the size of the demand that will be imposed on the application under test in an execution. The metric unit used for defining 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 [16] [26] [20].

Workload is also defined by the distribution of load 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 [16] [26].

Workload modeling is the attempt to create a simple and general 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 [16] [26].

There are two kinds of workload models: descriptive and generative. The 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 [13].

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. 3). This model is applied to all the workload attributes, e.g., computation, memory usage, I/O behavior, communication, etc. [13]. Fig. 3 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.

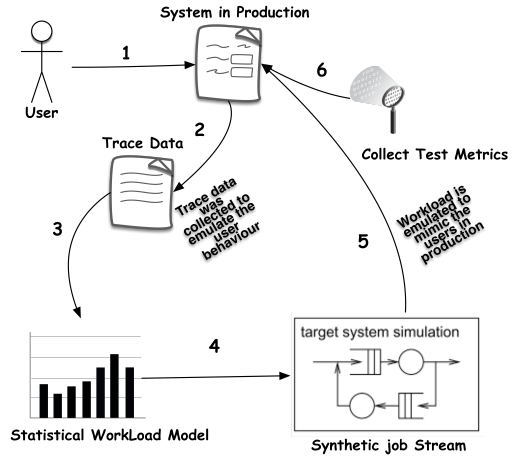


Figure 3: Workload modeling based on statistical data [13]

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 [13]. 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. 4).

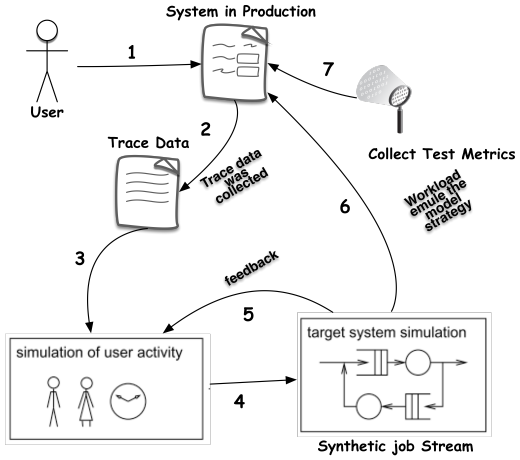


Figure 4: Workload modeling based on the generative model [13]

4. HYBRID METAHEURISTIC

A large number of researchers have recognized the advantages and huge potential of building hybrid mathematical programming methods and 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 [31] [7].

There are two main categories of metaheuristic combinations: collaborative combinations and integrative combinations. These are presented in Fig. 5 [33].

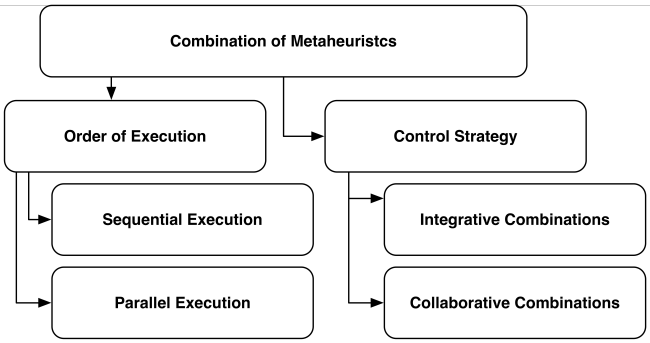


Figure 5: Categories of metaheuristc combinations [31]

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. The presented research work uses a type of collaborative combination with sequential execution.

5. IMPROVING LOAD, PERFORMANCE, AND STRESS SEARCH-BASED TESTING USING A HYBRID METAHEURISTIC APPROACH

The proposed solution makes it possible to create a generative model that evolves during the test. The proposed solution model uses genetic algorithms, Tabu search, and simulated annealing in two different approaches. 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. 6). The second approach uses the algorithms in a collaborative mode (hybrid metaheuristic). In this approach, the three algorithms share their best individuals found (Fig. 7).

The next subsections present details about the used metaheuristic algorithms (genotype representation and fitness function) and the IAdapter components.

5.1 Genotype representation

The genotype representation is composed by a linear vector with 23 genes. The first gene represents the name of an individual. The second gene represents the algorithm (genetic algorithm, simulated annealing, or Tabu search) used by the individual. The third gene represents the type of test (load, stress, or performance). The next genes 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.

Fig. 8 presents the genome 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.

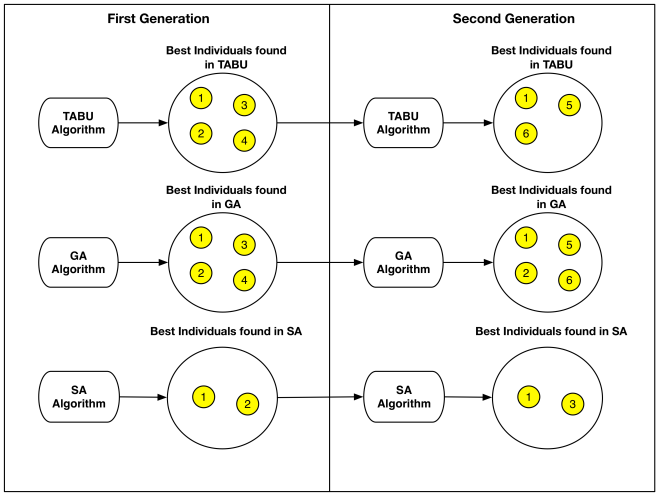


Figure 6: Use of the algorithms independently

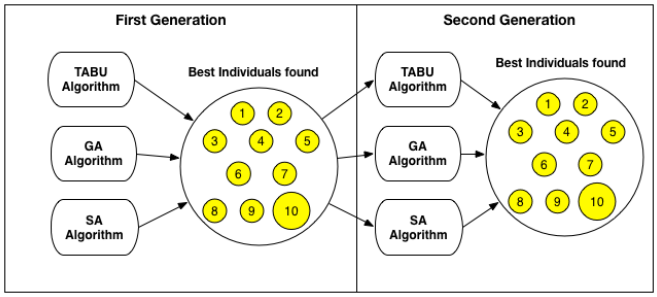


Figure 7: Use of the algorithms collaboratively

Fig. 9 shows the strategy used by the IAdapter tool to obtain the genotype of the neighbors for the Tabu search and simulated annealing algorithms. The neighbors are obtained by the modification of a single chromosome (scenario or number of users) in the genotype.

5.2 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.

5.3 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:

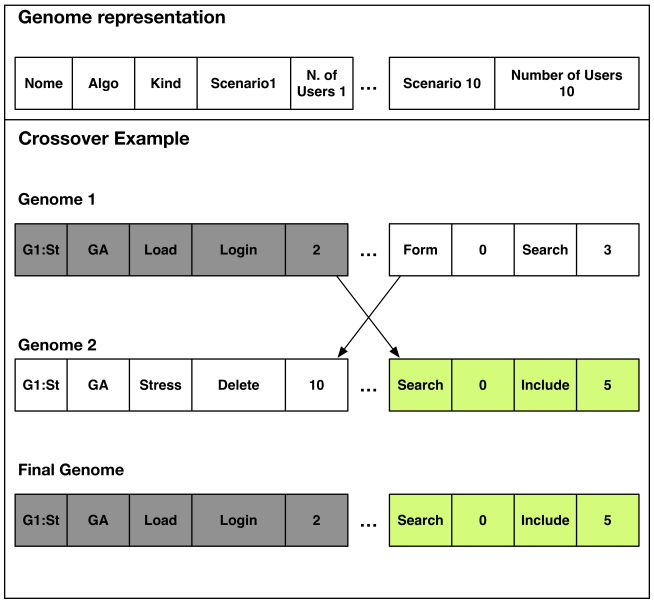


Figure 8: Genotype representation and crossover example

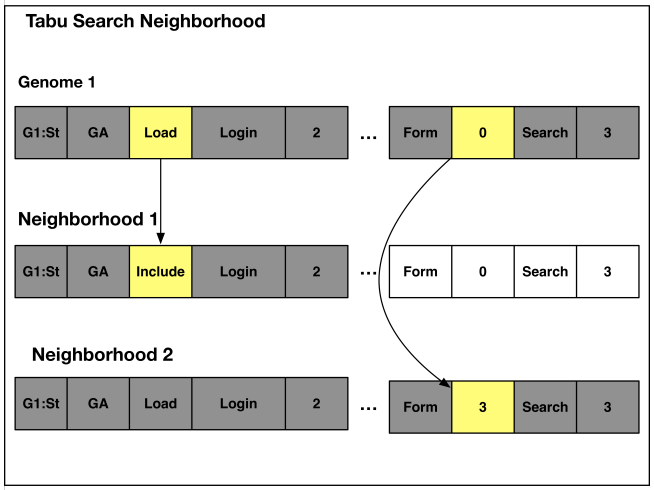


Figure 9: Tabu search and simulated annealing neighbor strategy

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

The proposed solution's fitness function uses a series of adaptable 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.

6. IADAPTER

IAdapter is a JMeter plugin designed to perform search-based load, performance, or stress tests. JMeter is a desktop application designed to test and measure the performance and functional behavior of applications [28]. The IAdapter plugin implements the solution proposed in Section 6.

JMeter has components organized in a hierarchical manner. The IAdapter plugin provides three main components: WorkLoadThreadGroup, WorkLoadSaver, and WorkLoadController.

WorkLoadThreadGroup is a component that creates an initial population and configures the algorithms used in IAdapter. Fig. 10 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.

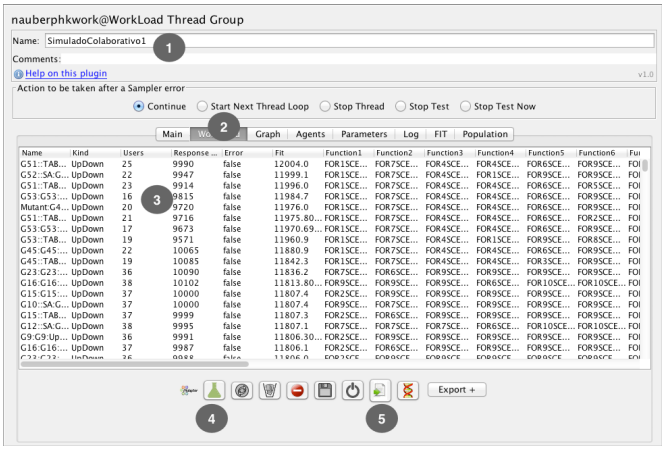


Figure 10: WorkLoadThreadGroup component

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 to the application under test and return the application to its original state.

7. 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 [35]. The main objective of search-based testing in performance, stress, and load tests 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 [35]. The application of evolutionary algorithms to load, performance, and stress tests involves finding the best- and worst-case execution times to determine whether timing constraints are fulfilled [2].

Some search-based tests use a cost (fitness) function to select the best individuals. There are two measurement units normally associated with the fitness function in a load, performance, or 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 [1] [37].

Table 1 shows a comparison between the presented research work and the research studies on load, performance, and stress tests presented by Afzal et al. [1]. Afzal's work adds to some of the latest research in this area ([17] [19] [11] [12] [4]).

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 divides 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.

Table 1: 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 Gois, 2015**
GA	Alander et al.,1998 [3] Wegener et al., 1996 and 199[39][22] Sullivan et al., 1998 [35] Briand et al., 2005 [8] Canfora et al., 2005 [9]	Wegener and Grochtmann 1998 [38] Mueller et al., 1998 [27] Puschner et al. [32] Wegener et al., 2000 [40] Gro et al., 2000 [21]	Di Penta, 2007 [29] Garoussi, 2006 [17] Garousi, 2008 [18] Garousi, 2010 [19]
Simulated Annealing (SA)			Tracey, 1998 [36]
Constraint Programming			Alesio, 2014 [12] Alesio, 2013 [11]
GA + Constraint Programming			Alesio, 2015 [4]
Customized Algorithm		Pohlheim, 1999 [30]	

The presented research work and Alesio's approach [4] use a hybrid approach with a functional tool. Table 2 presents the main differences between Alesio's and Gois'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 2: Main differences between Alesio's [4] and Gois's approaches

	Alesio et al. [4]	Gois et al.
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

8. EXPERIMENTS

Listing 1: SimulateConcurrentAcess 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   }
```

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:

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

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.

The first experiment implemented 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.

8.1 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 1).

Fig.11 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. 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 11: Best results obtained in 27 generations

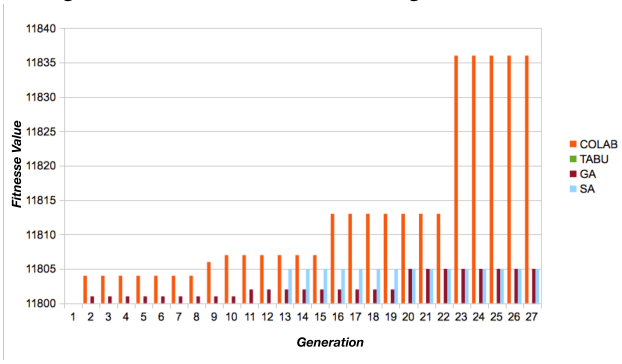


Table 3 presents the results obtained by the hybrid metaheuristic (HM) approach, genetic algorithm (GA), simulated annealing (SA), and Tabu search (TS) from 27 generations in the first experiment. The values are the maximum value of the fitness value obtained in each algorithm.

Table 3: 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

The signed-rank Wilcoxon non-parametrical procedure was used for comparing the results. The procedure showed that there was a significant improvement in the results with the collaborative approach.

8.2 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.
- 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 4 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 4: 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 5 shows an example of the six individuals with the highest fit 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 6 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.

9. CONCLUSION

This paper presented a hybrid metaheuristic approach for use in load, performance, and 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.

Table 5: 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 6: 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

The main contributions of this research are as follows: The presentation of a hybrid metaheuristic approach for use in load, performance, and stress tests; the development of a JMeter plugin for search-based tests; and the automation of the load, performance, or stress test execution 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.

10. REFERENCES

[1] A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976, 2009.

[2] Wasif Afzal, Richard Torkar, and Robert Feldt. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976, 2009.

[3] Jarmo T. JT Alander, Timo Mantere, and Pekka Turunen. Genetic Algorithm Based Software Testing. In *Neural Nets and Genetic Algorithms*, 1998.

[4] Stefano D I Alesio, Lionel C Briand, Shiva Nejati, and Arnaud Gotlieb. Combining Genetic Algorithms and Constraint Programming. *ACM Transactions on Software Engineering and Methodology*, 25(1), 2015.

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

[6] Cornel Barna, M Litoiu, and H Ghanbari. Autonomic load-testing framework. . . . *international conference on Autonomic . . .* , pages 91–100, 2011.

[7] Christian 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)*, 7505 LNCS(6):1–10, 2012.

[8] Lionel C. Briand, Yvan Labiche, and Marwa Shousha. Stress testing real-time systems with genetic algorithms. *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05*, page 1021, 2005.

[9] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. 2005., Canfora, G., An approach for QoS-aware service composition based on genetic algorithms.

[10] Microsoft Corporation. Performance Testing Guidance for Web Applications, November 2007.

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

[12] Stefano Di Alesio, Shiva Nejati, Lionel Briand, and Arnaud Gotlieb. Worst-Case Scheduling of Software Tasks – A Constraint Optimization Model to Support Performance Testing. *Principles and Practice of Constraint Programming*, pages 813–830, 2014.

[13] Giuseppe a. Di Lucca and Anna Rita Fasolino. Testing Web-based applications: The state of the art and future trends. *Information and Software Technology*, 48:1172–1186, 2006.

[14] 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.

[15] Iure De Sousa Fé and Pedro de Alcântara dos Santos. Os custos dos Testes de Desempenho e Estresse. 2004.

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

[17] Vahid Garousi. Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models using Genetic Algorithms. (August), 2006.

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

[19] Vahid Garousi. A Genetic Algorithm-Based Stress Test Requirements Generator Tool and Its Empirical Evaluation. *IEEE Transactions on Software Engineering*, 36(6):778–797, November 2010.

[20] Marcelo Canário 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.

[21] Hg Gross, Bryan F Jones, and David E Eyres. Structural performance measure of evolutionary testing applied to worst-case timing of real-time systems. *Software, IEE Proceedings-*, 147(2):25–30, 2000.

[22] B. Jones 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] ZM Jiang. *Automated analysis of load testing results*. PhD thesis, 2010.

[24] ZM Jiang and AE Hassan. Automated performance analysis of load tests. . . . , *2009. ICSM 2009. IEEE* . . . , 2009.

[25] Artur Luiz, Cunha Freitas, Orientadora Prof, and Renata Vieira. *Ontologias para Teste de Desempenho de Software*. PhD thesis, Pontifícia Universidade Católica do Rio Grande do Sul, 2011.

[26] Ian Molyneaux. *The Art of Application Performance Testing*. "O'Reilly Media, Inc.", January 2009.

[27] 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.

[28] Dmitri Nevedrov. Using JMeter to Performance Test Web Services. pages 1–11, 2007.

[29] Massimiliano Di Penta, Gerardo Canfora, and Gianpiero Esposito. Search-based testing of service level agreements. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1090–1097, 2007.

[30] Hartmut Pohlheim, Mirko Conrad, and Arne Griep. Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences. *Analysis*, (724):804—-814, 2005.

[31] Jakob 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*, 3562:41–53, 2005.

[32] 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.

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

[34] Corey Sandler, Tom Badgett, and TM Thomas. The Art of Software Testing. page 200, September 2004.

[35] Michael O Sullivan, Siegfried Vössner, Joachim Wegener, and Daimler-benz Ag. Testing Temporal Correctness of Real-Time Systems — A New Approach Using Genetic Algorithms and Cluster Analysis —. pages 1–20.

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

[37] Nigel James Tracey. *A search-based automated test-data generation framework for safety-critical software*. PhD thesis, Citeseer, 2000.

[38] J Wegener and M Grochtmann. Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Systems*, 15(3):275–298, 1998.

[39] Joachim Wegener, Harmen Sthamer, Bryan F Jones, and David E Eyres. Testing real-time systems using genetic algorithms. *Software Quality Journal*, 6(2):127–135, 1997.

[40] Harmen 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.