Improving 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.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 search-based testing is to find test scenarios that produce execution times that violate the timing constraints specified. The main purpose of this paper is determine if hybrid algorithms are superior to single metaheuristics when solving stress testing problems. The secondary objective of this paper is to improve Stress Test automation using a solution where the whole process of stress and performance tests, was carried out without the need for monitoring by a tester. The solution automatically selected the next scenarios to be run up to the limit of executions previously established. The research proposes a hybrid metaheuristic approach that uses genetic algorithms, simulated annealing, and Tabu search algorithms for use in 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. In the second experiment, the whole process of stress and performance tests, which took 3 days and about 1800 executions, was carried out without a tester. The tool automatically selected the next scenarios to be run up to the limit of six generations previously established.**

I. Introduction

Many systems must support concurrent access by hundreds or thousands of users. Failure to scale users 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, generated by system users, can result in highly critical failures and negatively affect the customers's 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 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 [4] [1].

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 on 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 (stress test)[3].

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.

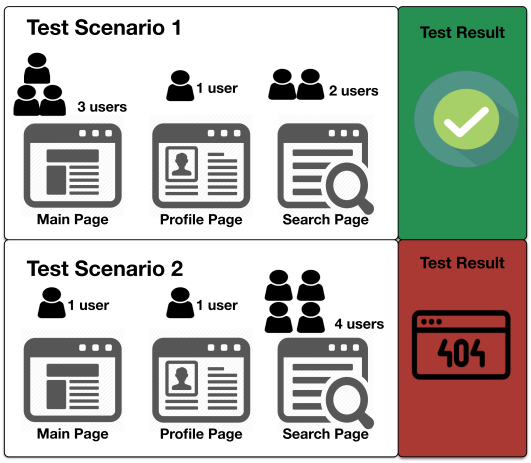
A stress test usually lasts for several hours or even a few days and only tests a limited number of workloads. The major challenge is to find the workloads that expose a major number of errors and to discover the maximum number of users supported by an application under test [6].

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

The research have two objectives:

- We wish to determine if hybrid algorithms are superior to single metaheuristics when solving stress testing problem

Figure 1: Possible test scenarios for a hypothetical application



and and improve the process of stress testing with a tool that evolves the test model during its execution.

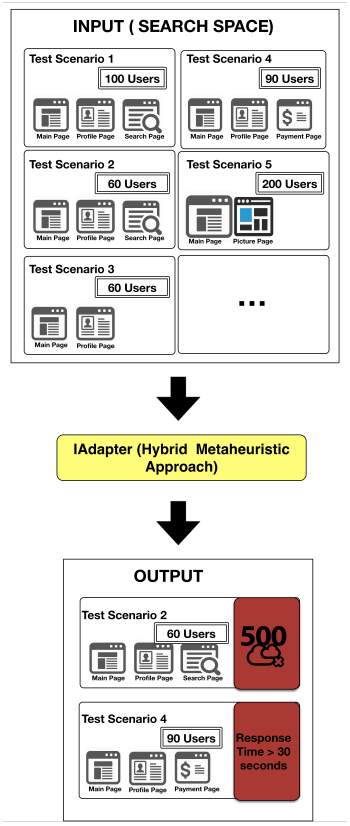
- We wish to improve stress test automation using a solution where the whole process of stress tests, was carried out without the need for monitoring by a tester. The solution automates the search for failure points of an application under test and automatically selected the next scenarios to be run up to the limit of executions previously established.

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

The 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. 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.

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 shows the results of two experiments performed using the IAdapter plugin. Section 8 discusses the related work. Conclusions and further work are presented in Section 9.

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



II. Load, Performance and Stress Test

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 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 [11]. The purpose of this test is to demonstrate that the system reaches its performance objectives [9].

In a load test, 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 test is the closest to real application use [2].

The stress test 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].

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 [2].

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 [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 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 [12] [2] [13].

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 [12] [2].

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 [12] [2].

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 [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. 3). This model is applied to all the workload attributes, e.g., computation, memory usage, I/O behavior, communication, etc. [11]. 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.

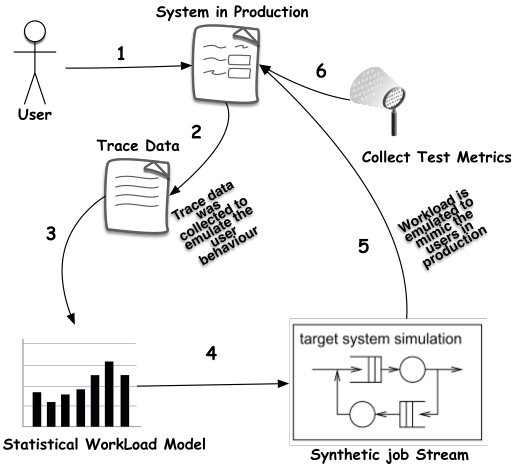


Figure 3: Workload modeling based on statistical data [11]

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. 4).

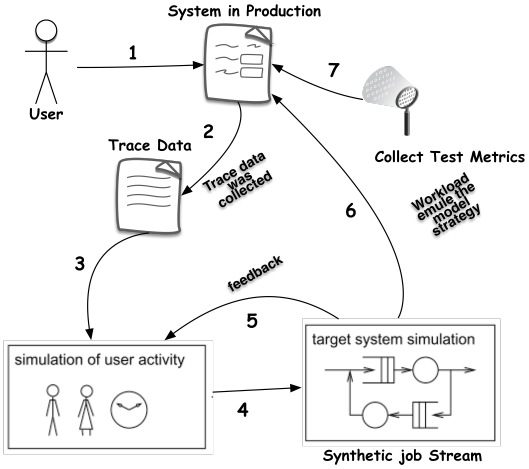


Figure 4: Workload modeling based on the generative model [11]

The presented research work uses a generative model.

IV. 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 [14] [15].

There are two main categories of metaheuristic combinations: collaborative combinations and integrative combinations. These are presented in Fig. 5 [16].

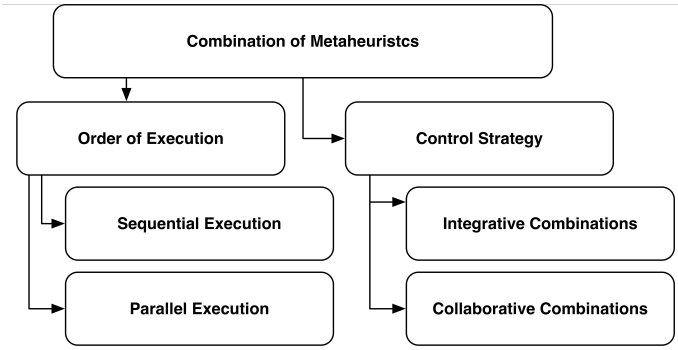


Figure 5: Categories of metaheuristc combinations [14]

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.

V. 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 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 [14].

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 three algorithms were selected because they can use a common genotype representation. 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, initial population and fitness function).

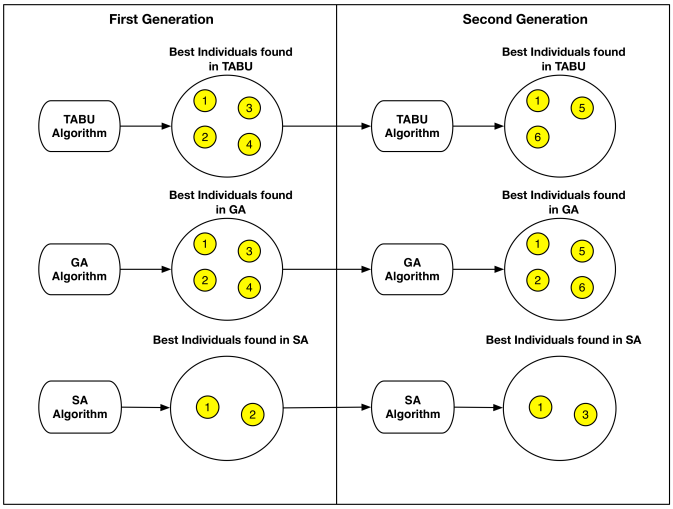


Figure 6: Use of the algorithms independently

A. 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,

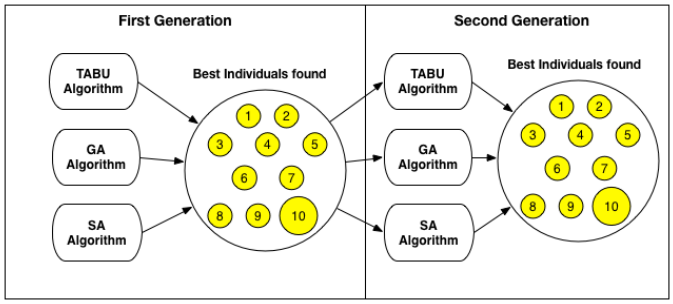


Figure 7: Use of the algorithms collaboratively

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.

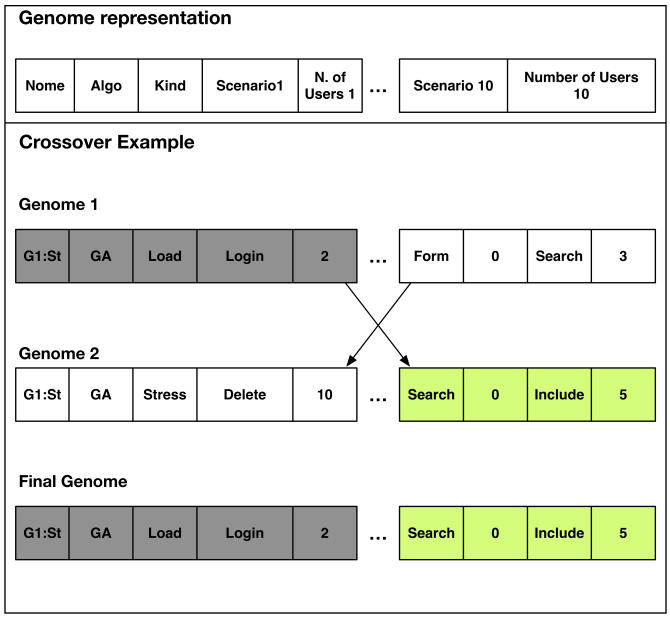


Figure 8: Genotype representation and crossover example

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.

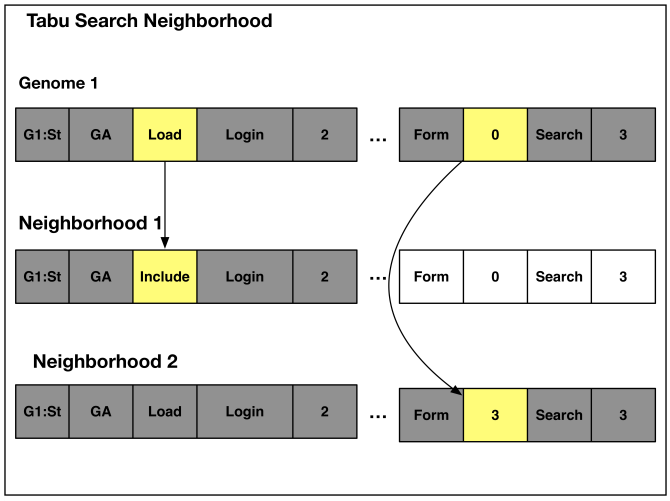


Figure 9: Tabu search and simulated annealing neighbor strategy

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.

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.

VI. IAdapter

IAdapter is a JMeter plugin designed to perform search-based stress tests. The plugin is available on www.iadapter.org. JMeter is a desktop application designed to test and measure the performance and functional behavior of applications. The IAdapter plugin implements the solution proposed in Section 5.

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

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

Fig. 10 presents the IAdapter Life Cycle. The main difference between IAdapter and JMeter tool is that the IAdapter provide a 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.

WorkLoadThreadGroup is a component that creates an initial population and configures the algorithms used in IAdapter. Fig. 11 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.

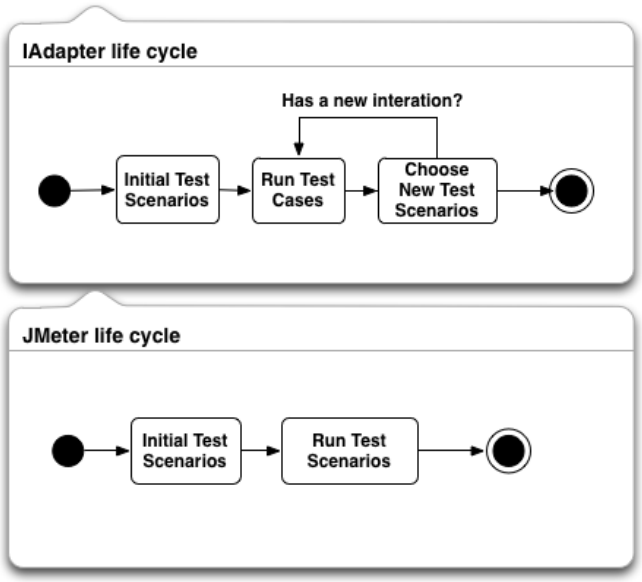


Figure 10: IAdapter life cycle

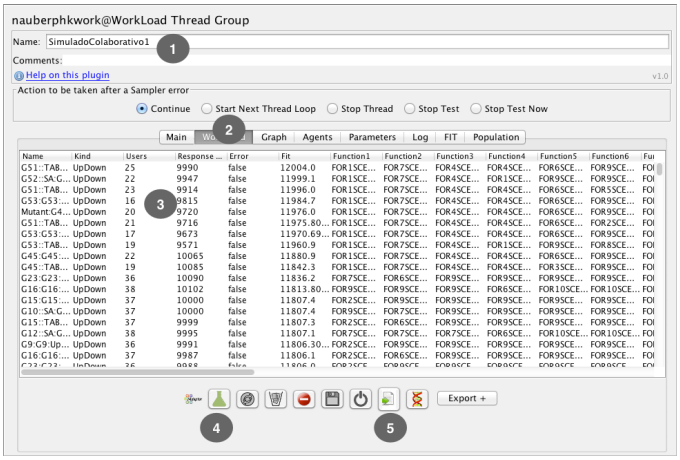


Figure 11: WorkLoadThreadGroup component

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.

VII. 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:

Listing 1: 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   }
```

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

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.

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 1).

Fig.12 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 12: Best results obtained in 27 generations

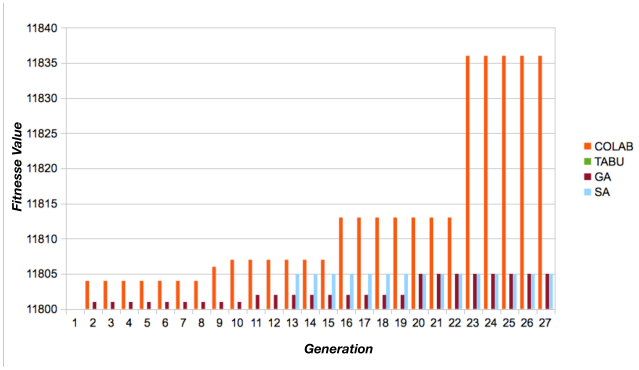


Table I presents the results obtained by the hybrid metaheuristic (HM) approach, genetic algorithm (GA), simulated annealing (SA), and Tabu search (TS) from 27 generations in the first experiment. The values are the maximum value of the fitness value obtained in each algorithm.

Table I: 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 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.
- 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 II 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 II: 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 III 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 IV 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 III: 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 IV: 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

VIII. 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 objective of search-based testing in stress 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 [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] [17].

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 V shows a comparison between the presented research work and the research studies on load, performance, and stress tests presented by Afzal et al. [18]. Afzal's work adds to some of the latest research in this area ([19] [4] [20] [21] [22]).

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 V: 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 [23] Wegener et al., 1996 a 1997 [24][25] Sullivan et al., 1998 [8] Briand et al., 2005 [26] Canfora et al., 2005 [27]	Wegener and Grochtmar 1998 [28] Mueller et al., 1998 [29] Puschner et al. [30] Wegener et al., 2000 [31] Gro et al., 2000 [32]	Di Penta, 2007 [33] Garoussi, 2006 [19] Garousi, 2008 [34] Garousi, 2010 [4]
Simulated Annealing (SA)			Tracey, 1998 [35]
Constraint Programming			Alesio, 2014 [21] Alesio, 2013 [20]
GA + Constraint Programming			Alesio, 2015 [22]
Customized Algorithm		Pohlheim, 1999 [36]	

Wegener et al. 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 [24].

Alander et al. 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 [23].

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 [25] [28] .

Gro et. al. 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 [32].

Tracey et al. 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 [35].

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[36].

Briand et al. 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 [26].

Di Penta et al. 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 [33].

Garousi presented a stress test methodology aimed at increasing chances of discovering faults related to distributed traffic in distributed systems. The technique uses as input a specified UML 2.0 model of a system, augmented with timing information.The results indicate that the technique is significantly more effective at detecting distributed traffic-related faults when compared to standard test cases based on an operational profile [19].

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 [22].

The presented research work and Alesio's approach [22] use a hybrid approach with a functional tool. Table VI 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 VI: Main differences between Alesio's [22] and IAdapter's approaches

	Alesio et al. [22]	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

IX. 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 load, performance, and 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 are a set 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. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2008.12.005

[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. [Online]. Available: http://books.google.com.br/books?id=GjyEFPkMCwcChttp://scholar.google.com/scholar?hl=en\&btnG=Search\&q=intitle:The+Art+of+Software+Testing\#0

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

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

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

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

[14] 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.

[15] 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. [Online]. Available: http://dx.doi.org/10.1016/j.asoc.2011.02.032

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

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

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

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

[20] 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.

[21] ——, "Worst-Case Scheduling of Software Tasks – A Constraint Optimization Model to Support Performance Testing," *Principles and Practice of Constraint Programming*, pp. 813–830, 2014. [Online]. Available: http://link.springer.com/10.1007/978-3-319-10428-7{_}58

[22] 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.

[23] J. T. J. Alander, T. Mantere, and P. Turunen, "Genetic Algorithm Based Software Testing," in *Neural Nets and Genetic Algorithms*, 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.9891\&rep=rep1\&type=pdfhttp://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.9891http://link.springer.com/chapter/10.1007/978-3-7091-6492-1_71

[24] 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

[25] 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. [Online]. Available: http://update-it.com/documents/eurostar1996.pdf

[26] 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.

[27] 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."

[28] 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. [Online]. Available: ⟨GotoISI⟩://WOS:000077759900004

[29] 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.

[30] 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.

[31] 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.

[32] 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. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=1504554

[33] 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.

[34] 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. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1389095.1389433

[35] N. J. Tracey, J. a. Clark, and K. C. Mander, "Automated Programme Flaw Finding using Simulated Annealing," 1998. [Online]. Available: http://kar.kent.ac.uk/21679/

[36] 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. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.6557{\&}amp;rep=rep1{\&}amp;type=pdf