RESEARCH

Improving Search-Based Stress Testing Using Q-Learning and a Hybrid Metaheuristic Approach

Nauber Gois[*†], Pedro Porfírio and André Coelho

[*]Correspondence:
naubergois@gmail.com
Departamento de Informática Aplicada, UNIFOR, Av. Washington Soares, 1321, Fortaleza, BR
Full list of author information is available at the end of the article
[†]Equal contributor

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. Performance degradation and consequent system failures usually arise in stressed conditions. Stress testing subjects the program to heavy loads. In this context, search-based testing is seen as a promising approach to verify timing constraints. The goal of this research was to use a reinforcement learning technique to optimize the choice of neighboring solutions to explore, reducing the time needed to obtain the scenarios with the longest response time in the application. A tool named IAdapter, a JMeter plugin used for performing search-based stress tests, was extended. One experiment was conducted. The experiment reveal that the best solutions found by the new Hybrid q-learning approach were on average 5.98% better than that achieved by our previous Hybrid approach without q-learning. The HybridQ algorithm consumed the same amount of time than Hybrid algorithm; however, it needs a much greater number of requests than that needed by other algorithms.

Keywords: Search-Based Test; Stress Testing; Hybrid metaheuristic; Q-Learning

1 Introduction

Many systems must support concurrent access to hundreds or thousands of users. Failure to provide scalable access to users may result in catastrophic failures and unfavorable media coverage [28].

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 is to fix them [31].

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 the concurrent or simultaneous access by several users can result in highly critical failures and negatively affect the customer's perception of the company [15] [28].

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 [18] [28].

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 [15].

The stress testing process in the industry still follows a non-automated and ad hoc model wherein the designer or tester is responsible for running the tests, analyzing the results, and deciding which new tests should be performed [29].

Typically, commercially available load test tools use test scripts, which are programs that test designers write to automate testing. These test scripts perform actions or mimic user actions on GUI objects of the system to feed input data. Current approaches to stress testing suffer from certain limitations. Their cost-effectiveness is highly dependent on the particular test scenarios that are used, and yet there is no support for choosing those scenarios. A poor choice of scenarios could lead to underestimating the system response time, thereby missing an opportunity to detect a performance problem [21].

Search-based testing is seen as a promising approach to verify timing constraints [1]. A common objective of a stress search-based test is to find scenarios that produce execution times that violate the specified timing constraints [40].

Stress tests need to occur within a delimited period in a software development project. The test period of a software can last from a few days to months, depending on the project schedule. During the test period, the tests need to find the highest number of application failures while consuming as little time as possible. Most research studies use search-based techniques to find the best- or worst-case test workloads. The presented research work is distinguished from others by improving the choice of neighboring solutions, thus reducing the time needed to obtain the scenarios with the longest response time in the application.

The present study extends the article “Improving stress search based testing using a hybrid metaheuristic approach” [20] to ascertain if the use of the Q-learning technique allows meta-heuristic algorithms to improve the search for application failures with a smaller number of requests while consuming a shorter amount of time, assuming that the same application can be submitted to more than one test execution. This paper addresses the following research question:

- Could the Q-learning technique be used to improve the choice of neighboring solutions, thus improving the number of requests and the time needed to find scenarios with the longest response time in the application under test?

The main contribution of this study is that it proposes the use of the Q-learning technique in the choice of neighboring solutions. One experiment was conducted to validate the proposed approach. The experiment was performed using an installed OpenCart application.

The remainder of the paper is organized as follows. The next section gives a background on metaheuristic, search-based stress testing and the Q-learning technique. Section 3 presents the related works to this study. Section 4 presents the hybrid approach proposed by Gois et al. [20]. Section 5 presents the proposed solution. Section 6 shows the results of the experiment performed using the HybridQ algorithm. Conclusions and further work are presented in Section 7.

2 Background

In computer science, “metaheuristic” is an accepted term for general techniques that are not specific to a particular problem. A metaheuristic is formally defined as an iterative

generation process that guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space [38].

Metaheuristics are strategies that guide the search process to efficiently explore the search space 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.

Several papers advocate the use of search-based algorithms as a good solution to generate good test cases, given the combinatorial nature of a software test [33] [8] [7] [6] [44] [35] [9]. Search-based software testing (SBST) is the process of automatically generating tests according to a test adequacy criterion using search-based optimization algorithms, which are guided by an objective fitness function. The role of the fitness function is to capture a test objective that, when achieved, makes a contribution to the desired test adequacy criterion. SBST commonly uses metaheuristics as the search algorithm [24].

Simulated annealing (SA) is a metaheuristic algorithm that tries to avoid being trapped in local optimal solutions by assigning probabilities to deteriorating moves. The SA procedure is inspired by the annealing process of solids. SA is based on a physical process in the metallurgy discipline or in solid matter physics. Annealing is the process of obtaining the low-energy states of a solid in heat treatment [27]. Tabu search (TS) is a metaheuristic that guides a local heuristic search procedure to explore the solution space beyond the local optimal and search with short-term memory to avoid cycles. It uses a tabu list to keep track of the last moves and to prevent retracing [19].

Genetic algorithms (GAs) are a metaheuristic algorithm based on concepts adopted from genetic and evolutionary theories. GAs consist of several components [25] [39]:

- a representation of the solution, referred to as the chromosome;
- the fitness of each chromosome, referred to as the objective function; and
- the genetic operations of crossover and mutation, which generate new offspring.

The crossover or recombination operation combines two or more individuals to produce new individuals. Mutation or modification operators lead to self-adaptation of individuals [10]. In search-based tests, the crossover operation creates two new test cases, namely, T1' and T2', by combining test cases from two pre-existing test cases: T1 and T2 [4]. Algorithm 1 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 operations, and the individuals for the next population are selected from the union of the old population and the offspring population [38].

Algorithm 1 Genetic Algorithm

```
1: s ← GenerateInitialSolution()
2: Evaluate(P)
3: while termination conditions not met do
4:     P1 ← Recombine(P)
5:     P2 ← Mutate(P1)
6:     Evaluate(P2)
7:     P ← Select(P2, P)
8: end while
```

A combination of one metaheuristic with components from other metaheuristics is called a hybrid metaheuristic. The concept of hybrid metaheuristics, which combine 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 [38].

This paper addresses the use of hybrid metaheuristics in conjunction with reinforcement learning (RL) techniques in search-based tests. RL refers to both a learning problem and a subfield of machine learning. As a learning problem, it refers to learning to control a system to maximize some numerical value that represents a long-term objective. The basic idea of RL is simply to capture the most important aspects of the real problem faced by a learning agent interacting with its environment to achieve a goal [41]. RL is learning what to do—how to map situations to actions—-to maximize a numerical reward signal. The learner needs to discover which actions yield the most reward by trying them [41].

In RL, an agent wanders in an unknown environment and tries to maximize its long-term return by performing actions and receiving rewards. The challenge is to understand how a current action will affect future rewards. A good way to model this task is with Markov decision processes (MDPs). MDPs provide a mathematical framework for modeling decision making. In RL, all agents act in two phases: exploration vs. exploitation. In the exploration phase, the agents try to discover better action selections to improve their knowledge. In the exploitation phase, the agents try to maximize their reward on the basis of what is already known.

One of the challenges that arise from RL is the trade-off between exploration and exploitation. To obtain a large reward, an RL agent must prefer actions that it has tried in the past and found to be effective in producing a reward. However, to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it already knows to obtain a reward, but it also has to explore to make better action selections in the future.

Q-learning is a model-free RL technique. It is a multi-agent learning algorithm that learns equilibrium policies in Markov games, just as it learns to optimize policies in MDPs [22].

Q-learning and related algorithms try to learn the optimal policy from its history of interaction with the environment. The history of an agent is a sequence of state-action-rewards:

$$< s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, r_4, s_4 .... >, \quad (1)$$

where s_n is a state, a_n is an action, and r_n is a reward.

In Q-learning, the system's objective is to learn a control policy $\pi = \sum_{n=0}^{\infty} \gamma^{\mathrm{n}} r_t + n$, where π is the discounted cumulative reward, γ is the discount rate (01), and r_t is the reward received after the execution of an action at time t. Figure 1 shows a summary version of the Q-learning algorithm. The first step is to generate the initial state of the MDP. The second step is to choose the best action or a random action based on the reward; hence, the actions with the best rewards are chosen.

3 Related Works

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 [40]. The use of search-based tests in stress tests is adequate because of the large number of combinations of scenarios involved and the limited period of time for testing. Only incrementing the number of users in a single chosen scenario may not be appropriate because

there may be errors that involve the use of more than one scenario simultaneously. The application of SBST algorithms for stress tests involves finding the best- and worst-case execution times (BCET and WCET, respectively) to determine whether the timing constraints are fulfilled [1].

There is a great difficulty in comparing the present work with some of the approaches presented in the state of the art owing to the unavailability of the tools used by each research. The present work intends to compare the proposed solution with other methods based on the use of single metaheuristics (namely, GAs, SA, and TS) and with an alternative hybrid approach that is based on the use of GAs jointly with constraint programming (CP).

In this section, the studies will be categorized by the unit of measure commonly used in fitness functions, the stage of development of the solution available, and the search approach used by each research study. There are two measurement units normally associated with the fitness function in a 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] [42].

Processor cycle measurement is deterministic in the sense that it is independent of the 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 nondeterministic approach; there is no guarantee of getting the same results for the same test inputs [1]. However, stress testing wherein testers have no access to the production environment should be measured by the execution time measurement [31] [1].

The solutions presented in the selected papers were classified into prototype or functional tool categories. A prototype is a draft version of a product that allows showing the intention behind a feature. A functional tool has all the features completely developed and a GUI interface that the end user can navigate. Table 1 shows a comparison between the research studies on load, performance, and stress tests. The columns represent the type of tool used (prototype or functional tool), and the rows represent the metaheuristic approach used by each research study (GA, TS, SA, or a customized algorithm). The table also sorts the research studies by the type of fitness function used (execution time or processor cycles).

The studies can be grouped into two main groups:

- Search-based stress testing on safety-critical systems.
- Search-based stress testing on non-safety-critical systems.

Safety-critical systems are real-time systems or critical mission systems such as medical treatment systems or aeronautical or automotive control systems. All other systems are in the non-safety-critical systems category.

3.1 Search-based stress testing on safety-critical systems

Search-based tests have great importance in domains such as avionics, automotive, and aerospace that feature safety-critical systems, whose failure could result in catastrophic consequences. The importance of software in such systems is permanently increasing owing to the need for a higher system flexibility. For this reason, the software components of these systems are usually subject to safety certification. In this context, software safety certification has to take into account the performance requirements, which specify the constraints on how the system should react to its environment and how it should execute on its hardware platform [13].

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 +Q-Learning			Our approach
GA + SA + Tabu Search			Gois et al., 2016 [20]
GA	Alander et al., 1998 [2] Wegener et al., 1996 and 1997 [46][26] Sullivan et al., 1998 [40] Briand et al., 2005 [11] Canfora et al., 2005 [12]	Wegener and Grochtmann, 1998 [45] Mueller et al., 1998 [32] Puschner et al. [37] Wegener et al., 2000 [47] Gro et al., 2000 [23]	Di Penta et al., 2007 [34] Garoussi, 2006 [16] Garousi, 2008 [17] Garousi, 2010 [18]
Simulated Annealing (SA)			Tracey, 1998 [43]
Constraint Programming			Di Alesio et al., 2014 [14] Di Alesio et al., 2013 [13]
GA + Constraint Programming			Di Alesio et al., 2015 [3]
Customized Algorithm		Pohlheim, 1999 [36]	

Usually, embedded computer systems have to fulfill real-time requirements. A faultless function of the systems depends not only on their logical correctness but also on their temporal correctness. Dynamic aspects such as the duration of computations, the memory actually needed during program execution, or the synchronization of parallel processes are of major importance for the correct function of real-time systems [26] .

The concurrent nature of some embedded software often makes unpredictable the order of external events triggering the systems tasks. Such an increase in software complexity renders performance analysis and testing increasingly challenging [13].

Reactive real-time systems must react to external events within the time constraints. Triggered tasks must execute within the deadlines. Shousha developed a methodology for the derivation of test cases that aims at maximizing the chance of critical deadline misses [39].

The main goal of search-based stress testing of safety-critical systems is to find a combination of inputs that cause the system to delay task completion to the greatest possible extent [39]. The followed approaches use metaheuristics to discover the worst-case execution times.

Wegener et al. [46] used GAs 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 microseconds [46]. Alander et al. [2] performed experiments in a simulator environment to measure the response time extremes of protection relay software using GAs. 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 [2].

Wegener and Grochtmann performed an experiment to compare GA with random testing. The fitness function used was the duration of the 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 those obtained by random testing [26] [45].

Gro et al. [23] presented a prediction model that 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 the BCET and WCET [23].

Briand et al. [11] used GA to find the sequence of arrival times of events for aperiodic tasks that would 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 the runs of the GA. Two case studies were conducted, and the results illustrated that RTTT is a useful tool to stress a system under test [11].

Pohlheim et al. used an extension of GAs 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].

Garousi presented a stress test methodology aimed at increasing the chances of discovering faults related to distributed traffic in distributed systems. The technique uses a specified UML 2.0 model as an input of a system, augmented with timing information. The results indicated that the technique is significantly more effective at detecting distributed traffic-related faults when compared to standard test cases based on an operational profile [16].

Di Alesio et al. described an approach based on CP to automate the generation of test cases that reveal, or are likely to reveal, task deadline misses. They evaluated it through a comparison with a state-of-the-art approach based on GAs. In particular, the study compared CP and GA in five case studies for efficiency, effectiveness, and scalability. The experimental results showed that, on the largest and most complex case study, CP performed significantly better than GA. The research proposes a tool-supported, efficient, and effective approach based on CP to generate stress test cases that maximize the likelihood of task deadline misses [13].

Di Alesio described stress test case generation as a search problem over the space of task arrival times. The research searched for worst-case scenarios maximizing deadline misses, where each scenario characterizes a test case. The research combined two strategies, GA and CP. The results showed that, in comparison with GA and CP in isolation, GA+CP achieved nearly the same effectiveness as CP and the same efficiency and solution diversity as GA, thus combining the advantages of the two strategies. Di Alesio concluded that a combined GA+CP approach to stress testing is more likely to scale to large and complex systems [3].

3.2 Search-based stress testing on non-safety-critical systems

Tracey et al. [43] used SA to test four simple programs. The results of the research showed that the use of SA was more effective with a larger parameter space. The authors highlighted the need for a detailed comparison of various optimization techniques to explore the WCET and BCET of the system under test [43].

Di Penta et al. [34] used GA to create test data that violated quality-of-service (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 that produced charts, applied the black-box

approach, with the fitness value calculated only on the basis of how close solutions violate QoS constraints. In the case of the audio workflow, the GA outperformed the random search. For the second case study, the use of the black-box approach successfully violated the response time constraint, showing the violation of QoS constraints for a real service available on the Internet [34].

Gois et al. proposed a hybrid metaheuristic approach using GAs, SA algorithm, and TS algorithm to perform stress testing. A tool named IAdapter, a JMeter plugin used for performing search-based stress tests, was developed. Two experiments were performed to validate the solution. In the first experiment, a signed-rank Wilcoxon nonparametric procedure was used for comparing the results. The significance 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 ran for 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 [20].

4 Improving Stress Search-Based Testing Using a Hybrid Metaheuristic Approach

This section presents the hybrid approach proposed by Gois et al. [20]. The solution proposed by Gois et al. makes it possible to create a model that evolves during the test. A plugin called IAdapter was implemented for the research. IAdapter is a JMeter plugin designed to perform search-based stress tests. The plugin is available at `www.github.com/naubergois/newiadapter`.

The proposed solution model uses GAs, TS, and SA in two different approaches. The current study initially investigated the use of these three algorithms. Subsequently, this paper will focus on other 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. 2). The second approach uses the algorithms in a collaborative mode (hybrid metaheuristic). In this approach, the three algorithms share their best individuals found (Fig. 3). The next subsections present the details about the used metaheuristic algorithms (representation, initial population, and fitness function).

4.1 Representation

The solution representation provides a common representation for all workloads. Each workload is composed of a linear vector with 21 positions (Figure 4 -❶). The first position represents a metadata with the name of an individual. The next positions represent 10 scenarios and their numbers of users (Figure 4 -❷).

The continuous vector representation was performed in order to simplify the implementation of the JMeter tool. Preliminary experiments were conducted, varying the number of scenarios per subject. From the results of these experiments, we came to the conclusion that 10 would be a reasonable number of scenarios for our purposes. The limit of 21 positions and 10 scenarios was used for the scenarios presented in the experiments but could be calibrated in new versions of the plugin. 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 4 presents the solution representation and an example using the crossover operation. In the example, solution 1 (Figure 4 -❸) has the Login scenario with two users, the Search scenario with four users, the Include scenario with one user, and the Delete scenario with two users. After the crossover operation with solution 2 (Figure 4 -❹), we obtain a solution with the Login scenario with two users, the Search scenario with four users, the Update scenario with three users, and the Include scenario with five users (Figure 4 -❺). Figure 4 -❻ shows the strategy used by the proposed solution to obtain the neighbors for the TS and SA algorithms. The neighbors are obtained by the modification of a single position (scenario or number of users) in the vector.

4.2 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 into 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 relate 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.

4.3 Objective (fitness) function

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

$$\begin{aligned} fit = {} & numberOfUsersWeight * numberOfUsers \\ & - 90percentileweight * 90percentiletime \\ & - 80percentileweight * 80percentiletime \\ & - 70percentileweight * 70percentiletime \\ & - maxResponseWeight * maxResponseTime \\ & - penalty \end{aligned} \tag{2}$$

The users and response time factors were chosen because they are common units of measurement in load test tools [31]. 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 the response time of an application under test runs longer than the service level. The penalty is calculated by the following equation:

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

5 Improving Stress Search-Based Testing Using Q-Learning and a Hybrid Metaheuristic Approach

The goal of this research was to use a RL technique to optimize the choice of neighboring solutions to explore, thus reducing the time needed to obtain the scenarios with the longest response time in the application. The research assumes as premise that the same application under performance tests can be submitted to more than one cycle of tests execution, reducing the cost of the exploration phase of the q-learning algorithm used.

The solution (i.e., HybridQ) uses the GA, SA, and TS algorithms in a collaborative approach. Just like most RL problems, the proposed solution works in two different phases: exploration and exploitation. The following subsections show the details of the exploration and exploitation phases and the integration between metaheuristics and the Q-learning algorithm.

5.1 Exploration phase

The exploration phase uses a Markov model, as shown in Fig. 5; the proposed MDP model has three main states based on the response time. A test may have a response time greater than 1.2 times the maximum response time allowed, between 0.8 and 1.2 times the maximum response time allowed, or less than 0.8 times the maximum response time allowed. The values of 1.2 and 0.8 were chosen on the basis of the assumption of a tolerance margin of 20% for the application under test. This margin may be higher or lower, depending on the business requirements of the application.

The algorithm maintains three different tables (Table 2), one for each state. The selection of which table to use depends on the response time of the application.

Algorithm 2 Exploration phase table selection

```
1: if responseTime < 0.8 * maxResponseTime  then
2:     return qTableBellowServiceLevel
3: end if
4: if responseTime >= 0.8 * maxResponseTime and responseTime <= 1.2* maxResponseTime then
5:     return qTableServiceLevel
6: end if
7: if responseTime > 1.2 * maxResponseTime then
8:     return qTableAboveServiceLevel
9: end if
```

Algorithm 3 shows the main steps of the exploration phase. The possible actions in the MDP are the change in one of the test scenarios and an increase or decrease in the number of users. In line 1, the algorithm chooses a random action (increase, decrease, or maintain the number of users). In line 2, the algorithm chooses a random test scenario. In lines 3 to 7, the algorithm checks if there exists a q value for the pair (action and test scenario); if none exists, then a zero value is assigned. In line 8, the algorithm checks if the new solution increases the fitness value. A solution receives a positive reward when an action increases the fitness value and a negative reward when an action reduces the fitness value. Finally, the algorithm updates the qTable with the new q value.

Unlike in the traditional approach, updating of the Q values for each action also occurs in the exploitation phase. The exploration phase ends when no value of Q equals zero. Table 2 presents hypothetical Q values for a test. In Table 2, it can be observed that the algorithm in the Service Level state are in the exploitation phase because there is no other value of Q that is equal to zero.

Algorithm 3 HybridQ exploration phase

```
1: action ← Random.createAction()
2: testScenario ← Random.chooseTestScenario()
3: if qTable.containsKey(action + "#" + testScenario) then
4:     qValue ← qTable.get(action + "#" + testScenario);
5: else
6:     qValue ← 0
7: end if
8: if newSolution.getFitness() > oldSolution.getFitness() then
9:     qValue ← ReinforcementLearning.alpha * reward + (1 − ReinforcementLearning.alpha) * qValue
10: else
11:     qValue ← ReinforcementLearning.alpha * −reward + (1 − ReinforcementLearning.alpha) * qValue
12: end if
13: qTable.update(action + "#" + testScenario, qValue)
```

Table 2: Hypothetical MDP Q values

Above Service Level	**Scenario 1**	**Scenario 2**
Increment Users	0.2	0.0
Reduce Users	0.1	0.2
Phase	Exploration	Exploration
Service Level	**Scenario 1**	**Scenario 2**
Increment Users	0.2	0.11
Reduce Users	0.1	-0.2
Phase	Exploitation	Exploitation
Below Service Level	**Scenario 1**	**Scenario 2**
Increment Users	0.0	0.2
Reduce Users	0.1	0.0
Phase	Exploration	Exploration

5.2 Exploitation phase

The main objective of the exploitation phase is to choose the best neighboring solution on the basis of the Q value. The research expected that Q-learning improves the hybrid algorithm proposed by Gois et al. by replacing the random characteristic of the TS, SA, and GA operators with the direction given by the Q-learning technique in the exploration phase. Algorithm 4 presents the main steps of the exploitation phase. In the first line, the algorithm gets the original encoded solution. In lines 2 to 11, HybridQ gets the maximum, the second maximum, or the third maximum q value, depending on the random value of the random variable. The algorithm chooses one of the three largest values of q. The variation of the highest values was inserted in the algorithm to escape the local optima. In line 12, the algorithm gets the key value in the table that has the maximum q value. In line 13, the key is separated into two parts using the # delimiter. The first part of the key is the action, and the second part is the test scenario. If the action is equal to the '*up*' value, the number of encoded solution users is incremented. If the action is equal to the '*down*' value, the number of encoded solution users is decremented. Finally, the test scenario is changed and the new encoded solution is returned.

5.3 Integration between metaheuristics and the Q-learning algorithm

The Q-learning algorithm is used by TS or SA algorithm to obtain the neighbors and in the mutation operation of the GA. Unlike in the traditional processes of obtaining neighboring solutions such as random change and permutation, the decision to change a encoded solution gene is made from the action that has the highest value of Q. Fig. 6 shows how one of the neighbors of a test is generated using Q-learning in IAdapter. The solution uses a service called Q-Neighborhood Service to generate the neighbor from the action that has the highest value of Q.

Algorithm 4 HybridQ exploitation phase

```
Gene[]encodedsolution ← service.getTestencodedsolution()
random ← Random.nextInt(3)
if random==1 then
    qMaxValue ← qTable.getMaxValue(responseTime)
end if
if random==2 then
    qMaxValue ← qTable.getSecondMaxValue(responseTime)
end if
if random==3 then
    qMaxValue ← qTable.getThirdMaxValue(responseTime)
end if
key ← qTable.selectKey(qMaxValue)
String[] keySplit ← key.split('#')
action ← keySplit[0]
testScenario ← keySplit[1]
if action=='up' then
    increaseUsers(encodedsolution)
end if
if action=='down' then
    decreaseUsers(encodedsolution)
end if
encodedsolutionPosition ← Random.nextInt(encodedsolution.length)
changeTestScenario(encoded solution,testScenario,encoded solutionPosition)
```

6 Experiment

We conducted one experiment to verify the effectiveness of the HybridQ algorithm. The iterated racing procedure (irace) was applied as an automatic algorithm configuration tool for tuning the metaheuristics parameters. Iterated racing is a generalization of the iterated F-race procedure to automate the arduous task of configuring the parameters of an optimization algorithm [30]. The best parameters obtained from irace were a population size of five individuals, a crossover value of 0.7551, a mutation value of 0.7947, an elitism value of 0.5356, and a maximum number of iterations of 16. The experiment ran for 16 generations in a docker environment on a server with 16 GB of memory and 500 GB of hard disk space. The experiment used an initial population of five individuals by metaheuristics. The GA used the top four individuals from each generation in the crossover operation. The tabu list was configured with a size of 10 individuals and expired every two generations. The mutation operation was applied to 79% of the population on each generation. The experiments used TS, GAs, SA, the hybrid metaheuristic approach proposed by Gois et al. [20], and the HybridQ approach.

The objective function applied was intended to maximize the response time of the scenarios being tested. In the experiment, better fitness values coincided with finding scenarios with higher values of the response time. A penalty was applied when the response time was greater than the maximum response time expected. The experiment used the following fitness (goal) function:

$$\begin{aligned} fitness = {} & 20 * 90percentiletime \\ & 20 * 80percentiletime \\ & 20 * 70percentiletime \\ & 20 * maxResponseTime \\ & -penalty \end{aligned} \tag{4}$$

For the experiment, an objective function with a single factor was chosen since users and response time are conflicting factors. All tests in the experiment were conducted without the need of a tester by automating the process of executing and designing the performance test scenarios.

6.1 Experiment research questions

The following research question is addressed:

- Does the Q-learning technique improve the choice of neighboring solutions, which can thus improve the number of requests and the time needed to find scenarios with the longest response time in the application under test?

6.2 Variables

The independent variable was the algorithms used in each experiment. The dependent variables were the optimal solution found by each algorithm, the number of requests to find the optimal solution, and the time of execution needed by each algorithm.

6.3 Hypotheses

- With regard to the optimal solution found by each algorithm:
  - $H1_0$ (null hypothesis) : The HybridQ algorithm did not find the best solution compared to the other metaheuristic approaches.
  - $H1_1$: The HybridQ algorithm found the best solution compared to the other metaheuristic approaches.
- With regard to the number of requests needed to find the optimal solution of each algorithm:
  - $H2_0$ (null hypothesis) : The HybridQ algorithm realizes more requests than the other algorithms in the experiments performed.
  - $H2_1$: The HybridQ algorithm does not realize more requests than the other algorithms in the executed experiments.
- With regard to the time consumed to find the optimal solution of each algorithm:
  - $H3_0$ (null hypothesis) : The HybridQ algorithm does not converge faster or in same time than the other algorithms in the experiments performed.
  - $H3_1$: The HybridQ algorithm converge faster or in same time than the other algorithms in the experiments performed.

6.4 Experiment phases

The experiment was conducted in two phases. The first phase verified the number of requisitions and the time required for the HybridQ exploration phase. The second phase ran the stress test using GA, TS, SA, the hybrid approach by Gois et al., and the HybridQ algorithm simultaneously.

6.5 OpenCart Experiment

The experiment was conducted to test the use of the HybridQ algorithm in a real implemented application. The chosen application was the OpenCart application, available at `opencart.com`. OpenCart is a free open-source ecommerce platform for online merchants. OpenCart works with PHP 5 and MySQL. The maximum tolerated response time in the test was 5 s. The whole process of stress and performance tests, which ran for 2 days

Table 3: Q values for the response times that were lower than the service level

Action	Feature	Q value	State	Feature	Q value
up	Main Page	-0.0405763	**up**	**Add to Cart**	**0.0390237**
down	Main Page	0.00079202	down	Add to Cart	-0.00079202
same	Main Page	-0.0398	same	Add to Cart	-0.0398
up	Search Page	-0.00079202	up	View Cart	-0.0398
down	Search Page	-0.0398	down	View Cart	-0.0398
same	Search Page	-0.0398	same	View Cart	-0.0398
up	Product Details	-0.00079202	up	Remove Item	-0.0398
down	Product Details	-0.00079202	down	Remove Item	-0.0398
same	Product Details	-0.0398	same	Remove Item	-0.0398

and with about 1500 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 11 generations previously established.

The experiment used the following application features:

- Main Page: The main page of the application.
- Search Item: The application searches for a product.
- Product Detail: The application shows the details about one product.
- Add to Cart: The application adds a product to the shopping cart.
- View Cart: The application displays the shopping cart.
- Remove Item: The application removes an item from the shopping cart.

6.5.1 Q-learning training phase

The application was submitted to 1 h of training with the Q-learning algorithm using all test scenarios; Table 3 shows the values of q obtained for response times lower than the service level. The action and state with the best q value were the increment in the number of users ('up') in the Add-to-Cart feature. The learning phase required 1431 requisitions for the application under test.

6.5.2 Results

Figure 7 presents the number of requests by the maximum fitness value. The HybridQ algorithm obtained the maximum fitness value : 364,860 ($H1_1$ hypothesis). It obtained a solution with a greater fitness value, but needed a much greater number of requests compared to the other algorithms, not contemplating hypothesis $H2_1$. SA was the algorithm that obtained the best fitness value with a minor number of requests (Fitness value of 175,980 on the first requests), followed by TS and GA ($H2_0$ hypothesis). TS and SA required two hours of tests, all other algorithms consumed the same amount of time of test (6 h) ($H3_0$ hypothesis). The scenario with the highest fitness value had 4.8 s of response time and 38 users:

- 25 users on the Search page;
- 10 users on the Add to Cart feature;
- 2 users removing items from the cart;
- 1 user on the Main Page.

The t-test and Wilcoxon rank sum test were applied using the R language. The test results showed that the HybridQ algorithm and the hybrid algorithm proposed by Gois et al. were superior to GA, TS, and SA, with $p < 0.02$. The t-test showed that the mean of the HybridQ fitness value was superior to that of the hybrid algorithm proposed by Gois et al.

```
 1:         Welch Two-Sample t-test
 2:
 3: data:  b\$MAXFIT and c\$MAXFIT
 4: t = 13.829, df = 31678, p-value < 2.2e-16
 5: alternative hypothesis: true difference in means is not equal to 0
 6: 95 percent confidence interval:
 7:  7506.846 9986.226
 8: sample estimates:
 9: mean of x mean of y
10:  332007.5  313260.9
```

6.6 Threats to validity

As a typical search strategy, it is also difficult to ensure that the execution times generated in the experiment represent the global optimum. In this work, we just evaluated the use of a single-objective algorithm. However, several multi-objective algorithms could be applied. An experiment was performed with the configuration obtained by the irace algorithm; however, new experiments are required to verify the sensitivity of the results. It is necessary to compare the current approach with the CP approaches presented in the state of art.

7 Conclusion

The present study extends the article "Improving stress search based testing using a hybrid metaheuristic approach" to ascertain if the use of the Q-learning technique allows metaheuristic algorithms to improve the search for application failures. One experiment was conducted to validate the proposed approach. The experiment used GAs, TS, SA, the hybrid approach proposed by Gois et al. [20], and the HybridQ algorithm. The experiment ran for 16 generations. The experiment used an initial population of five individuals by metaheuristics. All tests in the experiment were conducted without the need of a tester by automating the execution of the stress tests with the JMeter tool. HybridQ found the individuals with the greatest response time. The scenario with the greatest fitness value had 38 users on the Search Page, Add to Cart, Remove Item, and Main Page features. SA was the algorithm that obtained the best fitness value with a minor number of requests. TS and SA required two hours of tests, all other algorithms consumed the same amount of time of test (6 h). The results obtained can improve the decision-making process related to the service level definition for an application under test. In the experiment, the application supported only 38 users for the service level of 5 seconds. There is a range of future improvements in the proposed approach:

- More experimentation is also required to verify sensitivity of the results to hybrid metaheuristic configuration.
- The fitness approach of the Gois et al. solution is based on two conflicting measures: number of users and execution time. A multi-objective algorithm should be more adequate.
- It is necessary to compare the current approach with the constraint programming (CP) approaches presented in the state of art.

Competing interests
The authors declare that they have no competing interests.

List of abbreviations
HybridQ – Hybrid Algorithm with Q-Learning
SBST – Search-Based Test

Ethics approval and consent to participate
Not applicable

Consent for publication
All authors consent to the publication of the research

Availability of data and materials
The source code for the tool is available at github.com/naubergois/newiadapter

Funding
Not applicable

Author's contributions
All the authors contributed in the writing, research, and formulation of the paper

Endnotes
Not applicable

References
1. Afzal W, Torkar R, Feldt R (2009) A systematic review of search-based testing for non-functional system properties. Information and Software Technology 51(6):957–976,
2. Alander JTJ, Mantere T, Turunen P (1998) Genetic Algorithm Based Software Testing. Neural Nets and Genetic Algorithms
3. Alesio SDI, Briand LC, Nejati S, Gotlieb A (2015) Combining Genetic Algorithms and Constraint Programming. ACM Transactions on Software Engineering and Methodology 25(1)
4. Aleti A, Moser I, Grunske L (2016) Analysing the fitness landscape of search-based software testing problems. Automated Software Engineering pp 1–19,
5. Babbar C, Bajpai N, Sarmah D (2011) Web Application Performance Analysis based on Component Load Testing. International Journal of Technology
6. Baudry B, Fleurey F, Jézéquel JM, Le Traon Y (2002) Genes and bacteria for automatic test cases optimization in the. net environment pp 195–206
7. Baudry B, Fleurey F, Jézéquel JM, Le Traon Y (2005) Automatic test case optimization: A bacteriologic algorithm. ieee Software 22(2):76–82
8. Berndt DJ, Watkins A (2004) Investigating the performance of genetic algorithm-based software test case generation pp 261–262
9. Berndt DJ, Watkins A (2005) High volume software testing using genetic algorithms. In: System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on, IEEE, pp 318b–318b
10. Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computing Surveys 35(3):189–213,
11. Briand LC, Labiche Y, Shousha M (2005) Stress testing real-time systems with genetic algorithms. Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05 p 1021,
12. Canfora G, Penta MD, Esposito R, Villani ML (2005) An approach for QoS-aware service composition based on genetic algorithms. Proceedings of the 7th annual conference on Genetic and evolutionary computation
13. Di Alesio S, Nejati S, Briand L, Gotlieb A (2013) Stress testing of task deadlines: A constraint programming approach. IEEE Xplore pp 158–167,
14. Di Alesio S, Nejati S, Briand L, Gotlieb A (2014) Worst-Case Scheduling of Software Tasks – A Constraint Optimization Model to Support Performance Testing. Principles and Practice of Constraint Programming pp 813–830
15. Draheim D, Grundy J, Hosking J, Lutteroth C, Weber G (2006) Realistic load testing of Web applications. Conference on Software Maintenance and Reengineering (CSMR'06)
16. Garousi V (2006) Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models using Genetic Algorithms. Journal of Systems and Software (August)
17. Garousi V (2008) 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,
18. Garousi V (2010) A Genetic Algorithm-Based Stress Test Requirements Generator Tool and Its Empirical Evaluation. IEEE Transactions on Software Engineering 36(6):778–797,
19. Glover F, Martí R (1986) Tabu Search. ORSA Journal on computing pp 1–16
20. Gois N, Porfirio P, Coelho A, Barbosa T (2016) Improving Stress Search Based Testing using a Hybrid Metaheuristic Approach. Proceedings of the 2016 Latin American Computing Conference (CLEI) pp 718–728
21. Grechanik M, Fu C, Xie Q (2012) Automatically finding performance problems with feedback-directed learning software testing. 2012 34th International Conference on Software Engineering (ICSE) pp 156–166,
22. Greenwald A, Hall K, Serrano R (2003) Correlated Q-learning. Icml (3):84–89, URL http://www.aaai.org/Papers/Symposia/Spring/2002/SS-02-02/SS02-02-012.pdf
23. Gross H, Jones BF, Eyres DE (2000) Structural performance measure of evolutionary testing applied to worst-case timing of real-time systems. Software, IEE Proceedings- 147(2):25–30,
24. Harman M, McMinn P (2010) A theoretical and empirical study of search-based testing: Local, global, and hybrid search. IEEE Transactions on Software Engineering 36(2):226–247,
25. Hong TP, Wang HS, Chen WC (2000) Simultaneously applying multiple mutation operators in genetic algorithms. Journal of heuristics 6(4):439–455
26. J Wegener, K Grimm, M Grochtmann, H Sthamer BJ (1996) Systematic testing of real-time systems. EuroSTAR'96: Proceedings of the Fourth International Conference on Software Testing Analysis and Review

./images/qalgo.png

Figure 1: The Q-learning algorithm

27. Jaziri W (2008) Local Search Techniques: Focus on Tabu Search
28. Jiang Z (2010) Automated analysis of load testing results. PhD thesis, URL http://dl.acm.org/citation.cfm?id=1831726
29. Lewis WE, Dobbs D, Veerapillai G (2005) Software testing and continuous quality improvement. URL http://books.google.com/books?id=fgaBDd0TfT8C{\&}pgis=1
30. Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari TS (2016) The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3:43–58,
31. Molyneaux I (2009) The Art of Application Performance Testing: Help for Programmers and Quality Assurance. "O'Reilly Media, Inc.", URL http://books.google.com.br/books?id=TKUP3pxxsJsC
32. Mueller F, Wegener J (1998) A comparison of static analysis and evolutionary testing for the verification of timing constraints. Proceedings Fourth IEEE Real-Time Technology and Applications Symposium (Cat No98TB100245)
33. Pargas RP, Harrold MJ, Peck RR (1999) Test-data generation using genetic algorithms. Software Testing Verification and Reliability 9(4):263–282
34. Penta MD, Canfora G, Esposito G (2007) Search-based testing of service level agreements. Proceedings of the 9th annual conference on Genetic and evolutionary computation pp 1090–1097
35. Perumal K, Ungati J, Kumar G, Jain N, Gaurav R, Srivastava P (2011) Test data generation: a hybrid approach using cuckoo and tabu search. Swarm, Evolutionary, and Memetic Computing pp 46–54
36. Pohlheim H, Conrad M, Griep A (2005) Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences. SAE Technical Paper (724):804—-814,
37. Puschner P, Nossal R (1998) Testing the results of static worst-case execution-time analysis. Proceedings 19th IEEE Real-Time Systems Symposium (Cat No98CB36279)
38. Raidl GR, Puchinger J, Blum C (2010) Metaheuristic hybrids. In: Handbook of metaheuristics, Springer, pp 469–496
39. Shousha M (2003) Performance stress testing of real-time systems using genetic algorithms. PhD thesis, Carleton University Ottawa
40. Sullivan MO, Vössner S, Wegener J, Ag Db (1998) Testing Temporal Correctness of Real-Time Systems — A New Approach Using Genetic Algorithms and Cluster Analysis —. Proceedings of the 10th International Software Quality Week (QW'97), volume pp 1–20
41. Sutton RS, Barto AG (2012) Reinforcement learning. Learning 3(9):322, , 1603.02199
42. Tracey NJ (2000) A search-based automated test-data generation framework for safety-critical software. PhD thesis, Citeseer
43. Tracey NJ, Clark Ja, Mander KC (1998) Automated Programme Flaw Finding using Simulated Annealing. ACM SIGSOFT Software Engineering
44. Watkins A, Berndt D, Aebischer K, Fisher J, Johnson L (2004) Breeding software test cases for complex systems pp 10–pp
45. Wegener J, Grochtmann M (1998) Verifying timing constraints of real-time systems by means of evolutionary testing. Real-Time Systems 15(3):275–298,
46. Wegener J, Sthamer H, Jones BF, Eyres DE (1997) Testing real-time systems using genetic algorithms. Software Quality Journal 6(2):127–135, , URL http://www.springerlink.com/index/uh26067rt3516765.pdf
47. Wegener, Joachim and Pitschinetz, Roman and Sthamer H (2000) Automated Testing of Real-Time Tasks. Proceedings of the 1st International Workshop on Automated Program Analysis, Testing and Verification (WAPATV'00)

Figures

./images/independ.png

Figure 2: Use of the algorithms independently [20]

./images/collaborative.png

Figure 3: Use of the algorithms collaboratively [20]

./images/genomere.png

Figure 4: Solution representation, crossover, and neighborhood operators [20]

Figure 5: Markov Decision Process used by HybridQ

Figure 6: HybridQ NeighborHood Service

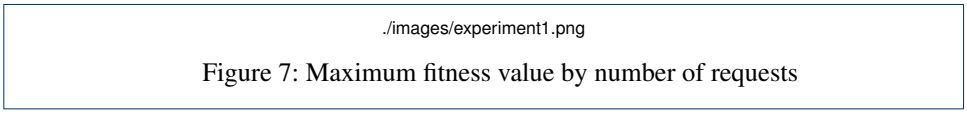

Figure 7: Maximum fitness value by number of requests