RESEARCH

Improving Search-Based Stress Testing using Q-Learning and 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. In this paper, We propose a hybrid metaheuristic approach that uses genetic algorithms, simulated annealing, and tabu search algorithms in a collaborative model using Q-Learning to improve stress search-based testing and automation. The main goal of the research is to find scenarios that maximize the number of users in the application with a response time below the response time service level. A tool named IAdapter, a JMeter plugin used for performing search-based stress tests, was developed. Two experiments were conducted to validate the proposed approach.

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 [24].

The explosive growth of the Internet has contributed to the increased need for applications that perform at an appropriate speed. Performance problems are often detected late in the application life cycle, and the later they are discovered, the greater the cost is to fix them [27].

The use of stress testing is an increasingly common practice owing to the fact that the increasing number of users. In this scenario, the inadequate treatment of a workload generated by concurrent or simultaneous access due to several users can result in highly critical failures and negatively affect the customer's perception of the company [11] [24].

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 [14] [24].

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 [11].

The stress testing process in the industry still follows a non-automated and ad-hoc model where the designer or tester is responsible for running the tests, analyzing the results and deciding which new tests should be performed [25].

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 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 system response time thereby missing an opportunity to detect a performance problem [17].

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 [36].

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 consuming as little time as possible. Most research studies use search-based techniques to find best- or worst-case test workloads. The presented research work is distinguished from others by improving the choice of neighboring solutions, 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" [16] in order to ascertain if the use of the Q-learning technique allows the meta-heuristic algorithms to find application failures with a smaller number of requests consuming a shorter 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, 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 Q-learning in the choice of neighboring solutions and replacing the mutation operator for genetic algorithms. Three experiments were conducted to validate the proposed approach. The first experiment was performed on an emulated environment, and the second one was performed using an installed JPetStore application.

The remainder of the paper is organized as follows. Section 2 presents a brief introduction about search-based testing. Section 3 presents concepts about search-based stress testing. Section 4 presents details about metaheuristic and hybrid metaheuristic. Section 5 presents concepts about Q-Learning. Section 6 presents the proposed solution. Section 7 shows the results of two experiments performed using the HybridQ algorithm. Conclusions and further work are presented in Section 8.

2 Background

In the computer science, the term metaheuristic is accepted for general techniques which are not specific to a particular problem. A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space [32].

Metaheuristics are strategies that guide the search process to efficiently explore the search space in order to find optimal solutions. Metaheuristic algorithms are approximate and usually non-deterministic and sometimes incorporate mechanisms to avoid getting trapped in confined areas of the search space.

Search-based testing (SBST) is the process of automatically generating tests according to a test adequacy criterion using search-based optimization algorithms, which are guided by a 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 [20].

Simulated Annealing (SA) is a metaheuristic algorithm that tries to avoid being trapped in local optimum solution by assigning probabilities to deteriorating moves. The SA procedure is inspired from the annealing process of solids. SA is based on a physical process in metallurgy discipline or solid matter physics. Annealing is the process of obtaining low energy states of a solid in heat treatment [23]. 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. Tabu Search uses a tabu list to keep track of the last moves, and prevents retracing [15].

Genetic Algorithms is a metaheuristic algorithm based on concepts adopted from genetic and evolutionary theories. GAs are comprised of several components [21] [33] :

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

The crossover operation or recombination recombines two or more individuals to produce new individuals. Mutation or modification operators causes a self-adaptation of individuals [6]. In Search-based tests, the crossover operator creates two new test cases 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 operators and the individuals for the next population are selected from the union of the old population and the offspring population [32].

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, combining different metaheuristic strategies and algorithms, dates back to the 1980s. Today, we can observe a generalized common agreement on the advantage of combining components from different search techniques and the tendency of designing hybrid techniques is widespread in the fields of operations research and artificial intelligence [32].

This paper addresses the use of hybrid metaheuristics in conjunction with reinforcement learning techniques in search-based tests.

Reinforcement learning (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 so as to maximize some numerical value which represents a long-term objective. The basic idea of Reinforcement learning is simply to capture the most important aspects of the real problem, facing a learning agent interacting with its environment to achieve a goal [37]. Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner needs to discover which actions yield the most reward by trying them [37].

In Reinforcement Learning, 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 (MDP). Markov decision processes (MDPs) provide a mathematical framework for modeling decision making. In Reinforcement Learning, all agents act in two phases: Exploration vs Explotation. In Exploration phase, the agents try to discover better action selections to improve its knowledge. In Exploitation phase, the agents try to maximize its reward, based on what is already know.

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

Q-learning is a model-free reinforcement learning technique. Q-learning is a multiagent learning algorithm that learns equilibrium policies in Markov games, just as Q-learning learns to optimize policies in Markov decision processes [18].

Q-learning and related algorithms try to learn the optimal policy from its history of interaction with the environment. A history of an agent is a sequence of state-action-rewards.Where s_n is a state, a_n is an action and r_n is a reward:

$$< s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, r_4, s_4.... >, \tag{1}$$

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 2 shows the summary version of 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 best rewards are chosen.

3 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 [36]. 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 (B/WCET) to determine whether timing constraints are fulfilled [1].

There is a great difficulty in comparing the present work with some of the approaches present in the state of the art, due to the lack of availability 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, genetic algorithms, simulated annealing, and Tabu Search), and with an alternative hybrid approach that is based on the use of genetic algorithms jointly with constraint programming.

In this section, the studies will be categorized by the unit of measure commonly used in fitnesse 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] [38].

Processor cycles measurement is deterministic in the sense that it is independent of a 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 [1]. However, stress testing where testers have no access to the production environment should be measured by the execution time measurement [27] [1].

The solutions presented in the selected papers was classified into prototype or functional tool categories. A prototype is a draft version of a product that allows you to show the intention behind a feature. A functional tool has all the features completely developed and a GUI interface to iterate with the end user. 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 (genetic algorithm, Tabu search, simulated annealing, or a customized algorithm). The table also sorts the research studies by the type of fitness function used (execution time or processor cycles).

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, 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 feature safety-critical systems, whose failure could result in catastrophic consequences The importance of software in such systems is permanently increasing due to the need of a higher system flexibility. For this reason, software components of these systems are usually subject to safety certification. In this context, software safety certification has to take into account performance requirements, specifying constraints on how the system should react to its environment, and how it should execute on its hardware platform [9].

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 [16]
GA	Alander et al.,1998 [2] Wegener et al., 1996 and 1997 [42][22] Sullivan et al., 1998 [36] Briand et al., 2005 [7] Canfora et al., 2005 [8]	Wegener and Grochtmann, 1998 [41] Mueller et al., 1998 [28] Puschner et al. [31] Wegener et al., 2000 [43] Gro et al., 2000 [19]	Di Penta et al., 2007 [29] Garoussi, 2006 [12] Garousi, 2008 [13] Garousi, 2010 [14]
Simulated Annealing (SA)			Tracey, 1998 [39]
Constraint Programming			Di Alesio et al., 2014 [10] Di Alesio et al., 2013 [9]
GA + Constraint Programming			Di Alesio et al., 2015 [3]
Customized Algorithm		Pohlheim, 1999 [30]	

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

The concurrent nature of some embedded software makes the order of external events triggering the systems tasks often unpredictable. Such an increase in software complexity renders performance analysis and testing increasingly challenging [9].

Reactive real-time systems must react to external events within time constraints. Triggered tasks must execute within deadlines. Shousha develops a methodology for the derivation of test cases that aims at maximizing the chance of critical deadline misses [33].

The main goal of Search-Based Stress testing of Safety-critical systems it is finding a combination of inputs that cause the system to delay task completion to the greatest possible extent [33]. The followed approaches use metaheuristics to discover the worst-case execution times.

Wegener et al. [42] 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 [42]. Alander et al. [2] 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 [2].

Wegener and Grochtmann performed an experiment to compare GA with random testing. The fitness function used was a 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 random testing [22] [41].

Gro et. al. [19] 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 [19].

Briand et al. [7] 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 [7].

Pohlheim et al. 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 [30].

Garousi presented a stress test methodology aimed at increasing 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 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 [12].

Di Alesio et al. describe an approach based on Constraint Programming (CP) to automate the generation of test cases that reveal, or are likely to, task deadline misses. They evaluate it through a comparison with a state-of-the-art approach based on Genetic Algorithms (GA). In particular, the study compares CP and GA in five case studies for efficiency, effectiveness, and scalability. The experimental results show that, on the largest and more complex case studies, CP performs significantly better than GA. The research proposes a tool-supported, efficient and effective approach based on CP to generate stress test cases that maximize the likelihood of task deadline misses [9].

Di Alesio describes 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 paper combines 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 [3].

3.2 Search-Based Stress Testing on Non Safety-critical systems

Tracey et al. [39] used simulated annealing (SA) to test four simple programs. The results of the research presented that the use of SA was more effective with a larger parameter space. The authors highlighted the need of a detailed comparison of various optimization techniques to explore WCET and BCET of the system under test [39].

Di Penta et al. [29] 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 that produces 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 the black-box approach successfully violated the response time constraint, showing the violation of QoS constraints for a real service available on the Internet [29].

Gois et al. proposes a hybrid metaheuristic approach using genetic algorithms, simulated annealing, and tabu search algorithms to perform stress testing. A tool named IAdapter, a JMeter plugin used for performing search-based stress tests, was developed. Two experiments were performed to validate the solution. In the first experiment, the signed-rank Wilcoxon non-parametrical procedure was used for comparing the results. The 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 1.800 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 [16].

4 Improving Stress Search Based Testing using Hybrid Metaheuristic Approach

This section presents the Hybrid approach proposed by Gois et al. [16]. 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 genetic algorithms, tabu search, and simulated annealing in two different approaches. The study initially investigated the use of these three algorithms. Subsequently, the study will focus 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. 3). The second approach uses the algorithms in a collaborative mode (hybrid metaheuristic). In this approach, the three algorithms share their best individuals found (Fig. 4). The next subsections present 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 by a linear vector with 21 positions (Figure 5 -❶). The first position represents an metadata with the name of an individual. The next positions represent 10 scenarios and their numbers of users (Figure 5 -❷). The fixed-length genome approach was chosen in reason of the ease of implementation in the JMeter tool. 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. 5 presents the solution representation and an example using the crossover operation. In the example, solution 1 (Figure 5 -❸) has the Login scenario with 2 users, the

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

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 in three ranges of values:

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

The percentages 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. 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 that the service level. The penalty is calculated by the follow 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 Hybrid Metaheuristic Approach

The goal of this research is 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. 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, named HybridQ, uses the GA, SA and Tabu Search algorithms in a collaborative approach. Just like most reinforcement learning problems the proposed solution works in two different phases: exploration and explotation.

5.1 Exploration phase

The exploration phase uses a markov model, as shown in Fig. 6, the proposed MDP model has three main states based on response time. A test may have a response time greater than 1.2 times the maximum response time alloowed, 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 from 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 exploration phase. The possible actions in MDP are the change of one of the test scenarios and an increase or decrease in the number of users. In line 1, the algorithm choose a random action (increase, decrease or maintain the number of users). In line 2, the algorithm choose one a random testScenario. In lines 3 to 7, the algorithm checks if there exists a q value for the pair (action and test scenario), if not exist a q value then zero value is assigned. In line 8, the algorithm checks i 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 the traditional approach, The update of Q values for each action also occurs in the exploitation phase. The exploration phase ends when no value of Q equals zero for a state, ie, unlike the traditional approach an agent belonging to one state may be in the exploration phase while another agent may be in the explotation phase. Table 2 presents hypothetical Q-values for a test. In Table 2, it can be observed that the agents in the Service Level state are in the exploitation phase because there is no other value of Q that equals 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	Explotation	Explotation
Bellow 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 based on the Q value.Algorithm 4 presents the main steps of exploitation phase. In first line, the algorithm gets the original genome. In second line, HybridQ gets the maximum q value.In third line, the algorithm gets the key value in Table that have the maximum q value. In line number 4, The key is separated into two parts using the # delimiter. The first part of the key is action and the second part is the test scenario. If the action equals '*up*' value, the genome is incremented in its users. If the action equals '*down*' value, the genome is incremented in its users.

Fig. 7 presents how one of the neighbors of a test is generated using Q-Learning. 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

```
1: Gene[]genome ← service.getTestGenome()
2: qMaxValue ← qTable.getMaxValue(responseTime)
3: key ← qTable.selectKey(qMaxValue)
4: String[] keySplit ← key.split('#')
5: action ← keySplit[0]
6: testScenario ← keySplit[1]
7: if action=='up' then
8:     increaseUsers(genome)
9: end if
10: if action=='down' then
11:     decreaseUsers(genome)
12: end if
13: genomePosition ← Random.nextInt(genome.length)
14: changeTestScenario(genome,testScenario,genomePosition)
```

6 Experiments

We conducted three experiments in order to verify the effectiveness of the HybridQ. The iterated racing procedure (irace) was applied as an automatic algorithm conguration tool for tunning metaheuristics parameters. Iterated racing is a generalization of the iterated F-race procedure to automatize the arduous task of configuring the parameters of an optimization algorithm [26]. The best parameters obtaind from irace was a population size of 5 individuals, a crossover value of 0.7551, a mutation value of 0.7947, an elitism value of 0.5356 and the maximum number of iterations of 16. All experiments ran for 16 generations in an emulated evinronment. The experiments used an initial population of 5 individuals by metaheuristics. The genetic algorithm used the top 4 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 79% of the population on each generation. The experiments use tabu search, genetic algorithms, simulated annealing, the hybrid metaheuristic approach proposed by Gois et al. [16] and the HybridQ approach.

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

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

For the experiments an objective function with a single factor was chosen, since users and response time are conflicting factors and will be approached together in a new multi-objective heuristic research. All tests in the experiment were conducted without the need of a tester, automating the process of executing and designing performance test scenarios.

6.1 Experiment Research Questions

The following research question is addressed:

- Is q-learning technique improve the choice of neighboring solutions, improving 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 is the algorithms used in each experiment. The dependent variables are: the optimal solution found by each algorithm, the number of requests to find optimal solution and the time of execution needed by each algorithm.

6.3 Hypotheses

- With regard to the optimal solution found by each algorithm:
  - $H1_0$ (A null hypothesis) : The HybridQ didn't found best solution than the other metaheuristic approaches.
  - $H1_1$: The HybridQ found best solution than the other metaheuristic approaches.
- With regard to the time consumed to find the optimal solution of each algorithm:
  - $H2_0$ (A null hypothesis) : The Hybrid algorithm consumed more requests than the other algorithms in the experiments performed.
  - $H2_1$: The HybridQ algorithm don't consume more requests than the other algorithms in the executed experiments.
- With regard to the the number of requests needed to find the optimal solution of each algorithm:
  - $H3_0$ (A null hypothesis) : The Hybrid algorithm consumed more time to find the optimal solution than the other algorithms in the experiments performed.
  - $H3_1$: The Hybrid algorithm don't consume more time to find the optimal solution than the other algorithms in the experiments performed.

6.4 Experiment phases

All the experiments was conducted in two phases. The first phase verified the number of requisitions and time required for the HybridQ exploration phase. The second phase ran the stress test using GA, Tabu Search, Simulated Annealing, Hybrid and HybridQ algorithms simultaneously.

6.5 The Ramp and Circuitous Treasure Experiment

The Ramp and Circuitous Treasure scenarios implement two performance antipatterns. Circuitous Treasure Hunt antipattern occurs when software retrieves data from a first componet, uses those results in a second component, retrieves data from the second component, and so on, until the last results are obtained (Fig. 10) [34] [35]. The Ramp is an antipattern where the processing time increases as the system is used. The Ramp can arise in several different ways. Any situation in which the amount of processing required to satisfy a request increases over time will produce the behavior. With the Ramp antipattern, the memory consumption of the application is growing over time (Fig. 11).

Fig. 8 presents the fitness value obtained by each metaheuristic. HybridQ metaheuristic obtained the better fitness values.

Despite having obtained the best fitness value in each generation, the Hybrid algorithm performs twice as many requests as the tabu search (Fig. 9). The HybridQ algorithm obtained the best fitness value. Figure 12 shows the average, minimal e maximum value by search method.

Fig. 13 presents the maximum, average, median and minimum fitness value by generation. The maximun fitness value increases at each generation. Figure 14 presents the density graph of the number of users by fitness value. The range between 100 and 150 users has the highest number of individuals found with higher fitness values.

Table 3 shows 4 individuals with 164 to 169 users. These are the scenarios with the maximum number of users found with the best response time. The first individual has 153 users in Happy Scenario 2, 16 users in Happy Scenario 1 and a response time of 13 seconds. None of the best individuals have one of the antipatterns used in the experiment.

Table 3: Best individuals found in the first experiment

Search Method	Generation	Users	fitness Value	Happy 2	Happy 1	Resp. Time
HybridQ	17	169	500740	153	16	13
HybridQ	16	169	500700	153	16	15
HybridQ	13	164	489740	149	15	13
HybridQ	15	164	489740	149	15	13

Fig. 15 presents the response time by the number of users of individuals with Happy Scenario 1 and Happy Scenario 2. The figure illustrates that the individuals with best fitness value have more users and lower response time.

Fig. 16 presents the Markov Decision Process (MDP) for the experiment. When the response time it is bellow or equal the service level, the action with major reward it is increase the number of users and include more positions with the Happy Scenario 2 (Happy 2). When the response time is greater than the service level, the action with the highest reward value decreases the number of users and includes more positions with Happy Scenario 2. The actions with the least reward value contains the both antipatterns Circuitous Treasure (CTH) and The Ramp antipatterns (Ramp).

In the first experiment, We conclude that the metaheuristics converged to scenarios with a happy path, excluding the scenarios with antipatterns. The hybridQ and hybrid metaheuristic returned individuals with higher fitness scores. However, the Hybrid metaheuristic made twice as many requests as Tabu Search to overcome it.

6.6 OpenCart Experiment

One 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 free open source ecommerce platform for online merchants. OpenCart works with PHP 5 and MySQL.

The maximum tolerated response time in the test was 10 seconds. The whole process of stress and performance tests, which run for 2 days and with about 1.800 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 eleven generations previously established.

The experiments use the follow application features:

- Main page: The main page of the application.
- Search item: The application searches a product.
- Product detail: The application shows detais about one item product.
- Add to Cart: The application adds a product to shopping cart.
- View Cart:The application displays the shopping cart.
- Remove Item: The application remove item from shopping cart.

6.7 JPetStore Application Experiment

One experiment was conducted to test the use of the HybridQ algorithm in a real implemented application. The chosen application was the JPetStore, available at `https://hub.docker.com/r/pocking/jpetstore/`. The maximum tolerated response time in the test was 500 seconds. Any individuals who obtained a time longer than the stipulated maximum time suffered penalties. The whole process of stress and performance tests, which run for 2 days and with about 1.800 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 eleven generations previously established. The experiments use the follow application features:

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

Table 4 presents the maximum number of users found in each scenario who obtained a time lower than the stipulated maximum time of 500 seconds.

Table 4: Maximum number of users of isolated test scenarios

Scenario	Max number of Users
Fish	85
Enter in The Catalog	60
Dogs	75
Cart	70
Register	90

The Fig. 17 and 18 present the response time by generation and by the number of users. The experiment used the following fitness function:

$$fitness = \begin{cases} n * \{3000 * numberOfUsers - 20 * 90percentiletime - 20 * 80percentiletime \\ -20 * 70percentiletime - 20 * maxResponseTime - penalty\}, \text{where n is a} \\ \text{bonus for an individual that have one or more chosen scenarios.} \end{cases} \quad (5)$$

The purpose of the fitness function is to maximize the number of users and minimize the response time in the tests containing a selected n functionalities. For example, it is possible to double the fitness value for tests that have the fish and user registration scenario.

This experiment tries to find the scenarios with a maximum number of users and best response time tests that contain the Cart and Register features. Figure. 19 and 20 shows the fitness value by generation. The HybridQ obtained the best fitness values in all generations.

The Fig. 21 shows the fitness value by number of requests by each Search Method. In the figure, it is possible to observe that HybridQ obtained the best fitness value with the same number of requests as the other algorithms.

Table 5 shows 4 individuals with 233 to 398 users. The first individual has 73 users in the Fish scenario, 17 users in the Dogs scenario, 50 users in the Cart scenario, 33 Users in the Register scenario and a response time of 357 seconds.

Table 5: Best individuals found in JPetStore first experiment

Search Method	Response	Users	Gen	Fitness	fish	Dogs	Cart	Register
HybridQ	357	173	9	44773	73	17	50	33
HybridQ	398	171	10	44831	57	33	48	33
HybridQ	331	164	9	44774	71	14	51	28
HybridQ	233	159	9	44783	63	31	32	33

We conclude that HybridQ found the individuals with greater number of users. The scenario with greater number of users is the Fish search feature. The hybrid metaheuristic with Q-Learning (HybridQ) returned individuals with higher fitness scores. The individual with best fitness value has 73 users in the Fish scenario, 17 users in the Dogs scenario, 50 users in the Cart scenario, 33 Users in the Register scenario and a response time of 357 seconds.

6.8 Threats to validity

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

7 Conclusion

Two experiments were conducted to validate the proposed approach. The experiments use genetic algorithms, tabu search, simulated annealing and an hybrid approach proposed by Gois et al. [16].

The experiments ran for 17 generations. The experiments used an initial population of 4 individuals by metaheuristics. All tests in the experiment were conducted without the need of a tester, automating the execution of stress tests with the JMeter tool.

In both experiments the HybridQ algorithm returned individuals with higher fitness scores. In the first experiment the metaheuristics converged to scenarios with a happy path, excluding the scenarios with the use of an antipatterns. The individual with the best fitness value has 64 users in the Happy Scenario 2, 81 users in the Happy Scenario 1 and a response time of 12 seconds. None of the best individuals have one of the antipatterns used in the experiment.

In the second experiment, HybridQ found the individuals with a greater number of users. The scenario with greater number of users is the Fish search feature. The hybrid metaheuristic with Q-Learning (HybridQ) returned individuals with higher fitness scores. The individual with the best fitness value has 73 users in the Fish scenario, 17 users in the Dogs scenario, 50 users in the Cart scenario, 33 Users in the Register scenario and a response time of 357 seconds. The use of HybridQ allowed the increase of more than 83 users when compared to the tests of isolated scenarios where a maximum of 90 users was achieved.

There is a range of future improvements in the proposed approach:

- Also as a typical search strategy, it is difficult to ensure that the execution times generated in the experiments represent 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.
- The fitness approach of the Gois et al. solution are based on two conflict 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 approaches presented in the state of art.

Among the future works of the research, new experiments should be performed comparing the proposed approach with the use of constraint programming. New research with multi-objective heuristic algorithms must be performed.

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. Elsevier B.V., vol 51, pp 957–976,
2. Alander JTJ, Mantere T, Turunen P (1998) Genetic Algorithm Based Software Testing. In: Neural Nets and Genetic Algorithms
3. Alesio SDI, Briand LC, Nejati S, Gotlieb A (2015) Combining Genetic Algorithms and Constraint Programming. vol 25
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
6. Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computing Surveys 35(3):189–213,
7. Briand LC, Labiche Y, Shousha M (2005) Stress testing real-time systems with genetic algorithms. p 1021,
8. Canfora G, Penta MD, Esposito R, Villani ML (2005) An approach for QoS-aware service composition based on genetic algorithms
9. Di Alesio S, Nejati S, Briand L, Gotlieb A (2013) Stress testing of task deadlines: A constraint programming approach. pp 158–167,
10. Di Alesio S, Nejati S, Briand L, Gotlieb A (2014) Worst-Case Scheduling of Software Tasks – A Constraint Optimization Model to Support Performance Testing. pp 813–830
11. Draheim D, Grundy J, Hosking J, Lutteroth C, Weber G (2006) Realistic load testing of Web applications. In: Conference on Software Maintenance and Reengineering (CSMR'06),
12. Garousi V (2006) Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models using Genetic Algorithms. August
13. Garousi V (2008) Empirical analysis of a genetic algorithm-based stress test technique. p 1743,
14. Garousi V (2010) A Genetic Algorithm-Based Stress Test Requirements Generator Tool and Its Empirical Evaluation. vol 36, pp 778–797,
15. Glover F, Martí R (1986) Tabu Search. pp 1–16
16. Gois N, Porfirio P, Coelho A, Barbosa T (2016) Improving Stress Search Based Testing using a Hybrid Metaheuristic Approach. In: Proceedings of the 2016 Latin American Computing Conference (CLEI), pp 718–728
17. Grechanik M, Fu C, Xie Q (2012) Automatically finding performance problems with feedback-directed learning software testing. Ieee, pp 156–166,
18. Greenwald A, Hall K, Serrano R (2003) Correlated Q-learning. 3, pp 84–89, URL http://www.aaai.org/Papers/Symposia/Spring/2002/SS-02-02/SS02-02-012.pdf
19. Gross H, Jones BF, Eyres DE (2000) Structural performance measure of evolutionary testing applied to worst-case timing of real-time systems. vol 147, pp 25–30,
20. Harman M, McMinn P (2010) A theoretical and empirical study of search-based testing: Local, global, and hybrid search. vol 36, pp 226–247,
21. Hong TP, Wang HS, Chen WC (2000) Simultaneously applying multiple mutation operators in genetic algorithms. Springer, vol 6, pp 439–455
22. J Wegener, K Grimm, M Grochtmann, H Sthamer BJ (1996) Systematic testing of real-time systems
23. Jaziri W (2008) Local Search Techniques: Focus on Tabu Search
24. Jiang Z (2010) Automated analysis of load testing results. PhD thesis, URL http://dl.acm.org/citation.cfm?id=1831726
25. Lewis WE, Dobbs D, Veerapillai G (2005) Software testing and continuous quality improvement. URL http://books.google.com/books?id=fgaBDd0TfT8C{\&}pgis=1
26. 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,
27. Molyneaux I (2009) The Art of Application Performance Testing: Help for Programmers and Quality Assurance, 1st edn. "O'Reilly Media, Inc."

./images/mdp1.png

Figure 1: Example of a simple MDP with three states and two actions

./images/qalgo.png

Figure 2: Q Learning algorithm

28. Mueller F, Wegener J (1998) A comparison of static analysis and evolutionary testing for the verification of timing constraints.
29. Penta MD, Canfora G, Esposito G (2007) Search-based testing of service level agreements. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp 1090–1097
30. Pohlheim H, Conrad M, Griep A (2005) Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences. 724, pp 804—-814,
31. Puschner P, Nossal R (1998) Testing the results of static worst-case execution-time analysis.
32. Raidl GR, Puchinger J, Blum C (2010) Metaheuristic hybrids. In: Handbook of metaheuristics, Springer, pp 469–496
33. Shousha M (2003) Performance stress testing of real-time systems using genetic algorithms. PhD thesis, Carleton University Ottawa
34. Smith C, Williams L (2002) Software Performance AntiPatterns; Common Performance Problems and their Solutions. vol 2, pp 797–806, URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.6968{\&}rep=rep1{\&}type=pdf
35. Smith CU, Williams LG (2003) More New Software Performance AntiPatterns: EvenMore Ways to Shoot Yourself in the Foot. pp 717–725, URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.4517{\&}rep=rep1{\&}type=pdf
36. 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 —. pp 1–20
37. Sutton RS, Barto AG (2012) Reinforcement learning. vol 3, p 322, , 1603.02199
38. Tracey NJ (2000) A search-based automated test-data generation framework for safety-critical software. PhD thesis, Citeseer
39. Tracey NJ, Clark Ja, Mander KC (1998) Automated Programme Flaw Finding using Simulated Annealing
40. Vetoio V (2011) PhD Thesis in Computer Science Automated generation of architectural feedback from software performance analysis results Catia Trubiani
41. Wegener J, Grochtmann M (1998) Verifying timing constraints of real-time systems by means of evolutionary testing. vol 15, pp 275–298,
42. Wegener J, Sthamer H, Jones BF, Eyres DE (1997) Testing real-time systems using genetic algorithms. vol 6, pp 127–135, , URL http://www.springerlink.com/index/uh26067rt3516765.pdf
43. Wegener, Joachim and Pitschinetz, Roman and Sthamer H (2000) Automated Testing of Real-Time Tasks

Figures

./images/independ.png

Figure 3: Use of the algorithms independently [16]

./images/collaborative.png

Figure 4: Use of the algorithms collaboratively [16]

./images/genomere.png

Figure 5: Solution representation, crossover and neighborhood operators [16]

./images/mdp3.png

Figure 6: Markov Decision Process used by HybridQ

./images/q-neighborservice.png

Figure 7: HybridQ NeighborHood Service

./images/experiment1-1.png

Figure 8: Fitness value obtained by Search Method

./images/experiment1-3.png

Figure 9: Number of requests by Search Method

./images/circuit.png

Figure 10: Circuitous Treasure Hunt sample [40]

./images/ramp.png

Figure 11: The Ramp sample [40].

./images/experiment1-4.png

Figure 12: Average, median, maximum and minimal fitness value by Search Method

./images/experiment1-5.png

Figure 13: Fitness value by generation

./images/experiment1-6.png

Figure 14: Density graph of number of users by fitness value

./images/experiment1-7.png

Figure 15: Response time by the number of users of individuals with Happy Scenario 1 and Happy Scenario 2

./images/mdpexperiment.png

Figure 16: Markov decision process of experiment with Circuitous Treasure and The Ramp antipatterns

./images/SearchSurface.png

Figure 17: Response time of individuals found in the experiment by search method

./images/searchsurface2.png

Figure 18: Response time of individuals found in the experiment by fitness value

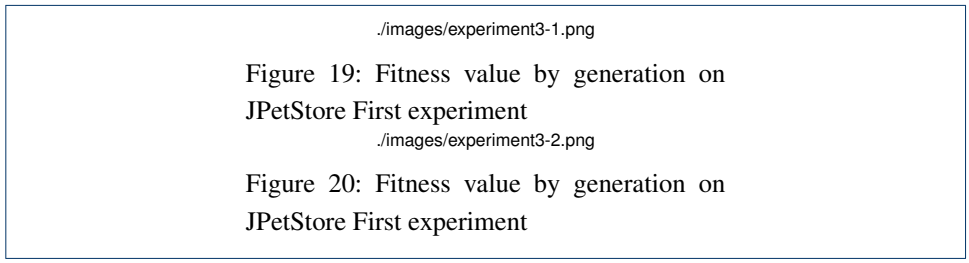

Figure 19: Fitness value by generation on JPetStore First experiment

Figure 20: Fitness value by generation on JPetStore First experiment

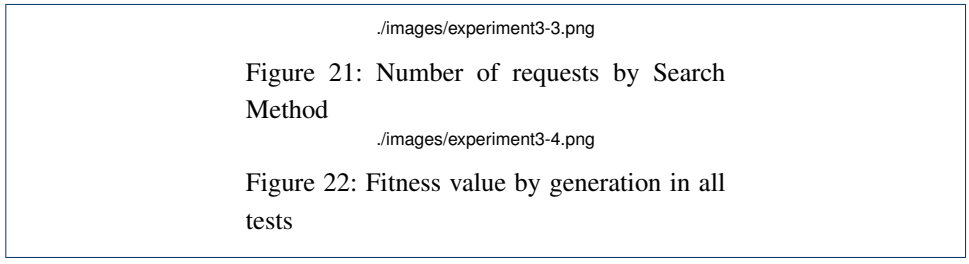

Figure 21: Number of requests by Search Method

Figure 22: Fitness value by generation in all tests