Noname manuscript No.
(will be inserted by the editor)

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

Nauber Gois · Pedro Porfírio · André Coelho

Received: date / Accepted: date

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, or stresses. In this context, search-based testing is seen as a promising approach for verifying timing constraints. In this paper, We propose an 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 by hundreds or thousands of users. Failure to providing scalable access to users may results in catastrophic failures and unfavorable media coverage [1].

The explosive growth of the Internet has contributed to the increased need for applications that perform at an appropriate speed. Performance problems are often

Nauber Gois, Pedro Porfírio, André Coelho
Av. Washington Soares, 1321 - Edson Queiroz, Fortaleza - CE, 60811-905
Tel.: +123-45-678910
Fax: +123-45-678910
E-mail: naubergois@gmail.com
E-mail: porfirio@unifor.br
E-mail: acoelho@unifor.br

detected late in the application life cycle, and the later they are discovered, the greater the cost to fix them [2].

The use of stress testing is an increasingly common practice owing to the increasing number of users. In this scenario, the inadequate treatment of a workload generated by concurrent or simultaneous access due to several users can result in highly critical failures and negatively affect the customers perception of the company [3] [1].

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

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

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 [6].

Typically, performance testing is accomplished using test scripts, which are programs that test designers write to automate testing. These test scripts performs actions or mimicking 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 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 [7].

Search-based testing is seen as a promising approach to verifying timing constraints [8]. A common objective of a Stress search-based test is to find scenarios that produce execution times that violate the specified timing constraints [9].

This research proposes extends the Hybrid Algorithm presented by Gois et al. [10]. The research approach uses Q-Learning reinforcement learning technique (HybridQ algorithm) to find the scenarios that maximize the number of users in the application with a response time below the response time service level. Two 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 features 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 Search-Based Testing

Search-Based Testing is the process of automatically generating test according to a test adequacy criterion using search-based optimization algorithms, which are guided by a 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 [11].

Search–Based Testing uses metaheuristic algorithms to automate the generation of test inputs that meet a test adequacy criterion. An advantage of meta-heuristic algorithms is that they are widely applicable to problems that are infeasible for analytic approaches [13] [12].

The application of metaheuristic search techniques to test case generation is a possibility which offers much benefits. Metaheuristic search techniques are high-level frameworks which utilise heuristics in order to find solutions to combinatorial problems at a reasonable computational cost. Such a problem may have been classified as NP-complete or NP-hard, or be a problem for which a polynomial time algorithm is known to exist but is not practical [14].

One of the most popular search techniques used in SBST belong to the family of Evolutionary Algorithms in what is known as Evolutionary Testing. Evolutionary Algorithms represent a class of adaptive search techniques based on natural genetics and Darwin's theory of evolution. They are characterized by an iterative procedure that works in parallel on a number of potential solutions to a problem. Figure 1 shows the cycle of an Evolutionary Algorithm when used in the context of Evolutionary Testing [13].

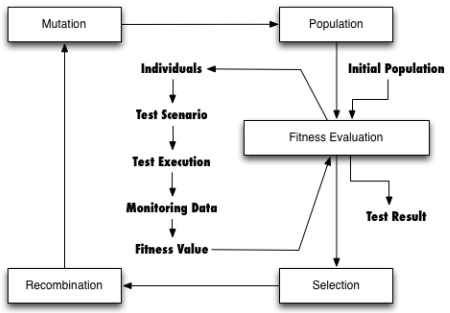


Fig. 1: Evolutionary Algorithm Search Based Test Cycle[13].

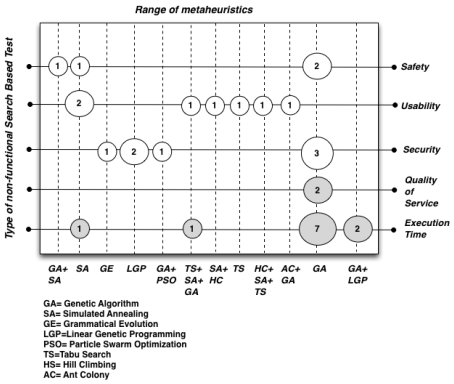


Fig. 2: Range of metaheuristics by Type of non-functional Search Based Test[8].

First, a population of possible solutions to a problem is created, usually at random. Starting with randomly generated individuals results in a spread of solutions ranging in fitness because they are scattered around the search–space. Next, each individual in the population is evaluated by calculating its fitness via a fitness function. The principle idea of an Evolutionary Algorithm is that fit individuals survive over time and form even fitter individuals in future generations. Selected individuals

are then recombined via a crossover operator. After crossover, the resulting offspring individuals may be subjected to a mutation operator. The algorithm iterates until a global optimum is reached or another stopping condition is fulfilled [13].

The fitness evaluation is the most time consuming task of SBST. However, for time consuming functional testing of complex industrial systems, minimizing the number of generated individuals may also be highly desirable. This might be done using an assumption about the "potential" of individuals in order to predict which individuals are likely to contribute to any future improvement. This prediction could be achieved by using information about similar individuals that have been executed in earlier generations.

2.1 Non-functional Search-Based Testing

SBST has made many achievements, and demonstrated its wide applicability and increasing uptake. Nevertheless, there are pressing open problems and challenges that need more attention like to extend SBST to test non-functional properties, a topic that remains relatively under-explored, compared to structural testing. There are many kinds of non-functional search based tests [8]:

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

A variety of metaheuristic search techniques are found to be applicable for non-functional testing including simulated annealing, tabu search, genetic algorithms, ant colony methods, grammatical evolution, genetic programming and swarm intelligence methods. The Fig. 2 shows a comparison between the range of metaheuristcs and the type of non-functional search based test.The Data comes from Afzal et al. [8]. Afzal's work adds to some of the latest research in this area ([15] [4] [16] [17] [18] [10]).

3 Search-Based Stress Testing

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 [9]. The application of SBST algorithms to stress tests involves finding the best- and worst-case execution times (B/WCET) to determine whether timing constraints are fulfilled [8].

There are two measurement units normally associated with the fitness function in stress test: processor cycles and execution time. The processor cycle approach describes a fitness function in terms of processor cycles. The execution time approach involves executing the application under test and measuring the execution time [8] [19].

Processor cycles measurement is deterministic in the sense that it is independent of system load and results in the same execution times for the same set of input parameters. However, such a measurement is dependent on the compiler and optimizer used, therefore, the processor cycles differ for each platform. Execution time measurement is a non deterministic approach, there is no guarantee to get the same results for the same test inputs [8]. However, stress testing where testers have no access to the production environment should be measured by the execution time measurement [2] [8].

Table 1 shows a comparison between the research studies on load, performance, and stress tests presented by Afzal et al. [8]. Afzal's work adds to some of the latest research in this area ([15] [4] [16] [17] [18] [10]). 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).

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			Gois et al. 2016 [10]
GA	Alander et al.,1998 [20] Wegener et al., 1996 and 1997 [21][22] Sullivan et al., 1998 [9] Briand et al., 2005 [23] Canfora et al., 2005 [24]	Wegener and Grochtmann, 1998 [25] Mueller et al., 1998 [26] Puschner et al. [27] Wegener et al., 2000 [28] Gro et al., 2000 [29]	Di Penta, 2007 [30] Garoussi, 2006 [15] Garousi, 2008 [31] Garousi, 2010 [4]
Simulated Annealing (SA)			Tracey, 1998 [32]
Constraint Programming			Alesio, 2014 [17] Alesio, 2013 [16]
GA + Constraint Programming			Alesio, 2015 [18]
Customized Algorithm		Pohlheim, 1999 [33]	

The studies can be grouped into two main groups:

- Search-Based Stress Tesing on Safety-critical systems.
- Search-Based Stress Testing on Industrial systems.

3.1 Search-Based Stress Tesing on Safety-critical systems

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 [16].

Usually, embedded computer systems have to fulfil 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, orthe synchronisation of parallel processes are of major importance for the correct function of real-time systems [22] .

The concurrent nature of embedded software makes the order of external events triggering the systems tasks is often unpredictable. Such increasing software complexity renders performance analysis and testing increasingly challenging. This aspect is reflected by the fact that most existing testing approaches target system functionality rather than performance [16].

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 [34].

The main goal of Search-Based Stress testing of Safety-critical systems it is finding a combination of inputs that causes the system to delay task completion to the greast extent possible [34]. The followed approaches uses metaheuristics to discover the worst-case execution times.

Wegener et al. [21] 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 [21]. Alander et al. [20] 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 [20].

Wegener and Grochtmann performed a experimentation to compare GA with random testing. The fitness function used was duration of execution measured in processor cycles. The results showed that, with a large number of input parameters, GA obtained more extreme execution times with less or equal testing effort than random testing [22] [25].

Gro et. al. [29] 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 [29].

Briand et al. [23] 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 [23].

Pohlheim and Wegener used an extension of genetic algorithms with multiple sub-populations, each using a different search strategy. The duration of execution measured in processor cycles was taken as the fitness function. The GA found longer execution times for all the given modules in comparison with systematic testing[33].

Garousi presented a stress test methodology aimed at increasing chances of discovering faults related to distributed traffic in distributed systems. The technique uses as input a specified UML 2.0 model of a system, augmented with timing information.The results indicate that the technique is significantly more effective at detecting distributed traffic-related faults when compared to standard test cases based on an operational profile [15].

Alesio, Nejati and Briand describe a 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, wthe 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 [16].

Alesio describe stress test case generation as a search problem over the space of task arrival times. The research search for worst case scenarios maximizing deadline misses where each scenario characterizes a test case. The paper combine two strategies, GA and Constraint Programming (CP). The results show that, in comparison with GA and CP in isolation, GA+CP achieves nearly the same effectiveness as CP and the same efficiency and solution diversity as GA, thus combining the advantages of the two strategies. Alesio concludes that a combined GA+CP approach to stress testing is more likely to scale to large and complex systems [18].

3.2 Search-Based Stress Testing on Industrial systems

Usually, the application of Search-Based Stress Testing on non safety-critical systems deals with the generation of test cases that causes Service Level Agreements violations.

Tracey et al. [32] used simulated annealing (SA) to test four simple programs. The results of the research presented that the use of SA was more effective with larger parameter space. The authors highlighted the need of a detailed comparison of various optimization techniques to explore WCET and BCET of the of the system under test [32].

Di Penta et al. [30] used GA to create test data that violated QoS constraints causing SLA violations. The generated test data included combinations of inputs. The approach was applied to two case studies. The first case study was an audio processing workflow. The second case study, a service producing charts, applied the black-box

approach with fitness calculated only on the basis of how close solutions violate QoS constraint. The genome representation is presented in Fig 3. The representation models a wsdl request to a webservice.

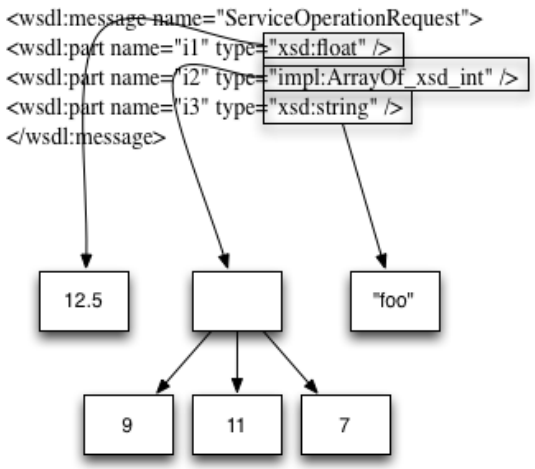


Fig. 3: Genome representation [30].

In case of audio workflow, the GA outperformed random search. For the second case study, use of black-box approach successfully violated the response time constraint, showing the violation of QoS constraints for a real service available on the Internet [30].

Gois et al. proposes an 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 significant level adopted was 0.05. The procedure showed that there was a significant improvement in the results with the Hybrid Metaheuristic approach. In the second experiment, the whole process of stress and performance tests, which took 3 days and about 1800 executions, was carried out without the need for monitoring by a test designer. The tool automatically selected the next scenarios to be run up to the limit of six generations previously established [10].

4 Metaheuristics and Hybrid Metaheuristics

In the computer science, the term metaheuristic is accepted for general techniques which are not specific to a particular problem. A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space [35].

Metaheuristics are strategies that guide the search process to efficiently explore the search space in order to find optimal solutions. Metaheuristic algorithms are approximate and usually non-deterministic and sometimes incorporate mechanisms to avoid getting trapped in confined areas of the search space. There are different ways to classify and describe metaheuristic algorithm [36]:

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

4.1 Single-Solution Based Metaheuristics

While solving optimization problems, single-solution based metaheuristics improve a single solution. They could be viewed as "walks" through neighborhoods or search trajectories through the search space of the problem at hand.

4.1.1 Neighborhood

The definition of Neighborhood is a required common step for the design of any Single-Solution metaheuristic (S-metaheuristic). The neighborhood structure it is a important piece in the performance of an S-metaheuristic. If the neighborhood structure is not adequate to the problem, any S-metaheuristic will fail to solve the problem. The neighborhood function N is a mapping: $N : S \rightarrow N^2$ that assigns to each solution s of S a set of solutions $N(s) \subset$ S [37].

The neighborhood definition depends representation associated with the problem. For permutation-based representations, a usual neighborhood is based on the swap operator that consists in swapping the location of two elements s_i and s_j of the permutation [37]. The Fig. 4 presents a example where a set of neighbors is found by permutation.

Single-Solution Based Metaheuristics methods are characterized by a trajectory in the search space. Two common S-metaheuristics methods are Simulated Annealing and Tabu Search.

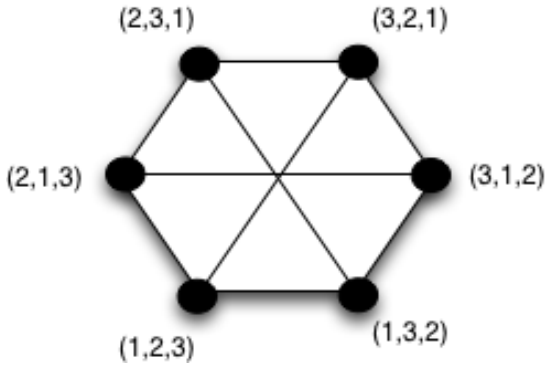


Fig. 4: An example of neighborhood for a permutation [37].

4.1.2 Simulated Annealing

Simulated Annealing (SA) is a randomized algorithm that tries to avoid being trapped in local optimum solution by assigning probabilities to deteriorating moves. The SA procedure is inspired from the annealing process of solids. SA is based on a physical process in metallurgy discipline or solid matter physics. Annealing is the process of obtaining low energy states of a solid in heat treatment [38].

The algorithmic framework of SA is described in Alg. 1. The algorithm starts by generating an initial solution in function *GenerateInitialSolution()*. The initial temperature value is determined in function *SetInitialTemperature()* such that the probability for an uphill move is quite high at the start of the algorithm. At each iteration a solution s_1 is randomly chosen in function *PickNeighborAtRandom(N(s))*. If s_1 is better than s, then s_1 is accepted as new current solution. Else, if the move from s to s_1 is an uphill move, s_1 is accepted with a probability which is a function of a temperature parameter *Tk* and s [35].

Algorithm 1 Simulated Annealing Algorithm

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

4.1.3 Tabu Search

Tabu Search (TS) is a metaheuristic that guides a local heuristic search procedure to explore the solution space beyond local optimal and search with short term memory to avoid cycles. Tabu Search uses a tabu list to keep track of the last moves, and don't allow going back to these [39].

The basic idea of TS is the explicit use of search history, both to escape from local minima and to implement a strategy for exploring the search space. A basic TS algorithm uses short term memory in the form of socalled tabu lists to escape from local minima and to avoid cycles [40].

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

Algorithm 2 Tabu Search Algorithm

$s \leftarrow GenerateInitialSolution()$
InitializeTabuLists(TL_1,...,TL_r)
while termination conditions not met **do**
    $N_a(s) \leftarrow \{s_1 \in N(s) | s_1$ does not violate a tabu condition, or it satisfies at least one aspiration condition $\}$
    $s_1 \leftarrow argmin\{f(s_2) | s_2 \in N_a(s)\}$
    UpdateTabuLists(TL_1,...,TL_r,s,s_1)
    $s \leftarrow s_1$
end while

4.2 Population-based metaheuristics

Population-based metaheuristics (P-metaheuristics) could be viewed as an iterative improvement in a population of solutions. First, the population is initialized. Then, a new population of solutions is generated. Finally, this new population is integrated into the current one using some selection procedures. The search process is stopped when a stopping criterion is satisfied. Algorithms such as Genetic algorithms (GA), scatter search (SS), estimation of distribution algorithms (EDAs), particle swarm optimization (PSO), bee colony (BC), and artificial immune systems (AISs) belong to this class of metaheuristics [41].

4.3 Genetic Algorithms

Genetic Algorithms could be a mean of solving complex optimization problems that are often NP Hard. GAs are based on concepts adopted from genetic and evolutionary theories. GAs are comprised of several components [42] [34] :

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

Algorithm 3 shows the basic structure of GA algorithms. In this algorithm, P denotes the population of individuals. A population of offspring is generated by the application of recombination and mutation operators and the individuals for the next population are selected from the union of the old population and the offspring population [35].

Algorithm 3 Genetic Algorithm

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

4.4 Hybrid Metaheuristics

However, in recent years it has become evident that the concentration on a sole metaheuristic is rather restrictive. A skilled combination of a metaheuristic with other optimization techniques, a so called hybrid metaheuristic, can provide a more efficient behavior and a higher flexibility when dealing with real-world and large-scale problems [43].

A combination of one metaheuristic with components from other metaheuristics is called a hybrid metaheuristic. The concept of hybrid metaheuristics has been commonly accepted only in recent years, even if the idea of combining different metaheuristic strategies and algorithms dates back to the 1980s. Today, we can observe a generalized common agreement on the advantage of combining components from different search techniques and the tendency of designing hybrid techniques is widespread in the fields of operations research and artificial intelligence [35].

There are two main categories of metaheuristic combinations: collaborative combinations and integrative combinations. These are presented in Fig. 5 [44].

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

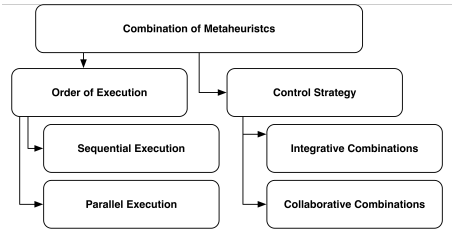


Fig. 5: Categories of metaheuristc combinations [45]

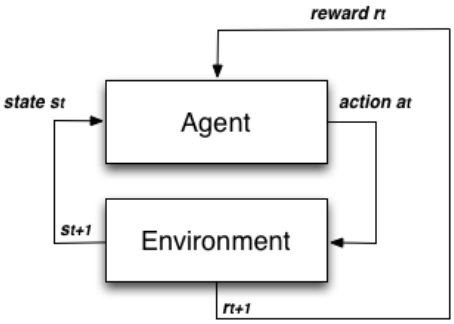


Fig. 6: Example of interation between some agent and the environment

One of the most popular ways of metaheuristic hybridization consists in the use of trajectory methods inside population-based methods. Population-based methods are better in identifying promising areas in the search space from which trajectory methods can quickly reach good local optima. Therefore, metaheuristic hybrids that can effectively combine the strengths of both population-based methods and trajectory methods are often very successful [35].

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

5 Reinforcement learning and Q-Learning

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. A typical setting where reinforcement learning operates is shown in Figure 6: A controller receives the controlled system's state and a reward associated with the last state transition. It then calculates an action which is sent back to the system.

The basis 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 [46]. Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner need to discover which actions yield the most reward by trying them [46].

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), which have become the dominant approach in Reinforcement Learning. There are two types of learning problems:

- Iteractive learning;
- Non-interactive learning.

In non-interactive learning, the natural goal is to find a good policy given a fixed number of observations. A common situation is when the sample is fixed. For example, the sample can be the result of some experimentation with some physical system that happened before learning started.

In Interactive learning, learning happens while interacting with a real system in a closed-loop fashion. A reasonable goal then is to optimize online performance, making the learning problem an instance of online learning. Online performance can be measured in different ways. A natural measure is to use the sum of rewards incurred during learning.

Interactive learning is potentially easier since the learner has the additional option to influence the distribution of the sample. However, the goal of learning is usually different in the two cases, making these problems incomparable in general.

In Reinforcement Learging, all agents act in two phases: Exploration vs Explotation. In Exploration phase, the agents tries discover better action selections to improve its knowledge. In Explotation phase, the agents tries to maximize its reward, based on what its already know.

One of the challenges that arise in reinforcement learning is the trade-off between exploration and exploitation. To obtain a lot of 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 reward, but it also has to explore in order to make better action selections in the future.

5.1 Markov decision processes

Markov decision processes (MDPs) provide a mathematical framework for modeling decision making. A countable MDP is defined as a triplet $M = (\chi, A, P_0)$ [47]. where χ is a set of states, A is a set of actions. The transition probability kernel P_0 assigns to each state-action pair $(x, a) \in \chi xA$

The six main elements of a MDP are:(1) state of the system, (2) actions, (3) transition probabilities, (4) transition rewards, (5) a policy, and (6) a performance metric [46].

The state of a system is a parameter or a set of parameters that can be used to describe a system. For example the geographical coordinates of a robot can be used to describe its state. A system whose state changes with time is called a dynamic system. Then it is not hard to see why a moving robot produces a dynamic system.

Actions are the controls allowed for an agent. Transition Probability denotes the probability of going from state i to state j under the influence of action a in one step. If an MDP has 3 states and 2 actions, there are 9 transition probabilities per action. Usually, the system receives an immediate reward ,which could be positive or negative, when it transitions from one state to another

A policy defines the learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. It corresponds to what in psychology would be

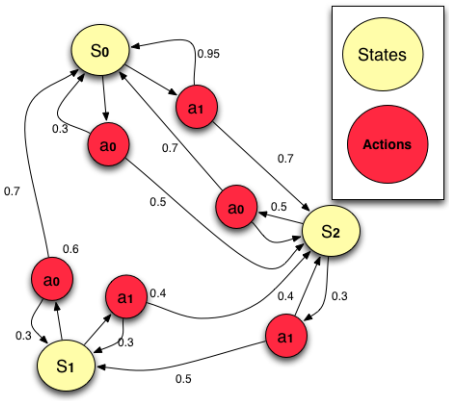


Fig. 7: Example of a simple MDP with three states and two actions

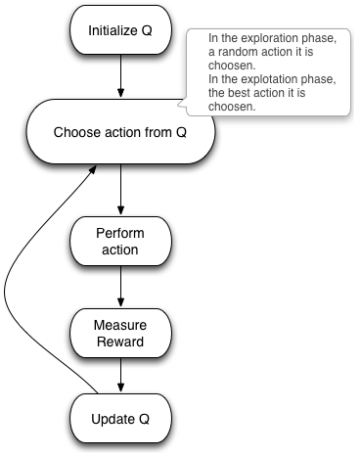


Fig. 8: Q Learning algorithm

called a set of stimulus–response rules or associations. Policys mapping from states to actions.

Performance Metric: Associated with any given policy, there exists a so-called performance metric — with which the performance of the policy is judged. Our goal is to select the policy that has the best performance metric.

5.2 Q-Learning

Q-learning is a model-free reinforcement learning technique. Q-learning, it is a multiagent learning algorithm that learns equilibrium policies in Markov games, just as Q-learning learns to optimal policies in Markov decision processes [48].

Q-learning and related algorithms tries 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 it is a state, a_n it 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 reiceved after execution an action at time t. The fig. 8 shows the summary version of Q-Learning algorithm. The first step it is to generate the initial state of the MDP. The second step it is to choose the best action or a random action based on the reward, the actions with best rewards are chosen.

6 Improving Stress Search Based Testing using Q-Learning and Hybrid Metaheuristic Approach

This section presents the Hybrid approach proposed by Gois et al.[10] and the HybridQ approach.

6.1 Hybrid Approach

A large number of researchers have recognized the advantages and huge potential of building hybrid metaheuristics. The main motivation for creating hybrid metaheuristics is to exploit the complementary character of different optimization strategies. In fact, choosing an adequate combination of algorithms can be the key to achieving top performance in solving many hard optimization problems [45] [49].

The solution proposed by Gois et al. [10] makes it possible to create a model that evolves during the test. The proposed solution model uses genetic algorithms, tabu search, and simulated annealing in two different approaches. The study initially investigated the use of these three algorithms. Subsequently, the study will focus in others Population-based and single point search metaheuristics. The first approach uses the three algorithms independently, and the second approach uses the three algorithms collaboratively (hybrid metaheuristic approach).

In the first approach , the algorithms do not share their best individuals among themselves. Each algorithm evolves in a separate way (Fig. 9).

The second approach uses the algorithms in a collaborative mode (hybrid metaheuristic). In this approach, the three algorithms share their best individuals found (Fig. 10). The next subsections present details about the used metaheuristic algorithms (Representation, initial population and fitness function).

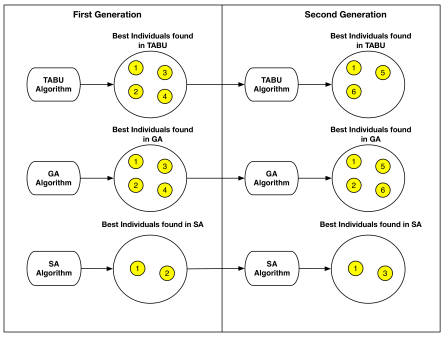


Fig. 9: Use of the algorithms independently [10]

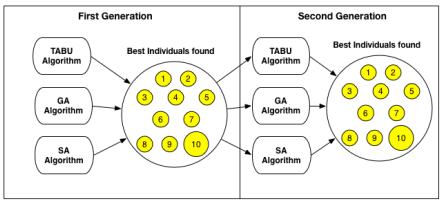


Fig. 10: Use of the algorithms collaboratively [10]

6.1.1 Representation

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

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

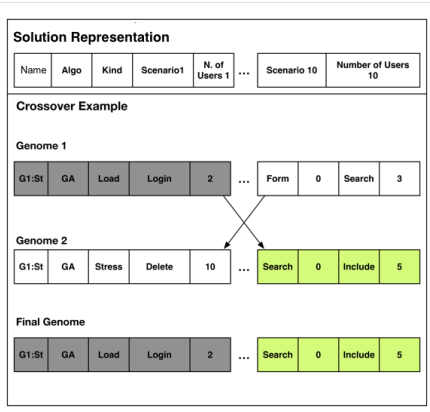


Fig. 11: Solution representation and crossover example [10]

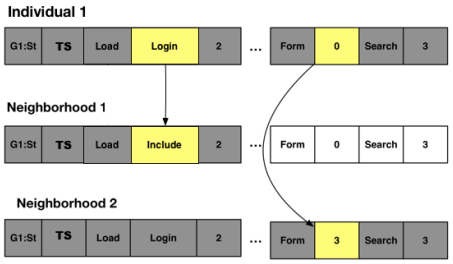


Fig. 12: Tabu search and simulated annealing neighbor strategy [10]

Fig. 12 shows the strategy used by the proposed solution to obtain the representation of the neighbors for the Tabu search and simulated annealing algorithms. The neighbors are obtained by the modification of a single position (scenario or number of users) in the vector.

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

6.1.3 Objective (fitness) function

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

$$\begin{aligned} fit = {} & numberOfUsersWeigth * numberOfUsers \\ & -90percentileweigth * 90percentiletime \\ & -80percentileweigth * 80percentiletime \\ & -70percentileweigth * 70percentiletime \\ & -maxResponseWeigth * maxResponseTime \\ & -penalty \end{aligned} \tag{2}$$

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

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

6.2 Hybrid Metaheuristic with Q-Learning Approach

The HybridQ algorithm uses the GA, SA and Tabu Search algorithms in a collaborative approach in conjunction with Q-Learning technique. The biggest difference between the Hybrid and HybridQ algorithms is the application of a series of modifications on individuals based on the Q-Learning algorithm before each generation.

Figure 14 shows the proposed MDP model for HybridQ. The 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. A test receives a positive reward when an action increases fitnesse value and a negative reward when an action reduces the fitnesse value. The possible actions in MDP are the change of one of the test scenarios and increase or decrease of the number of users.

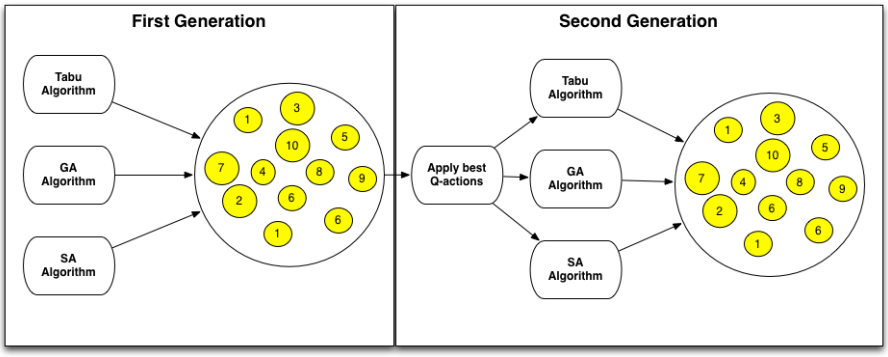
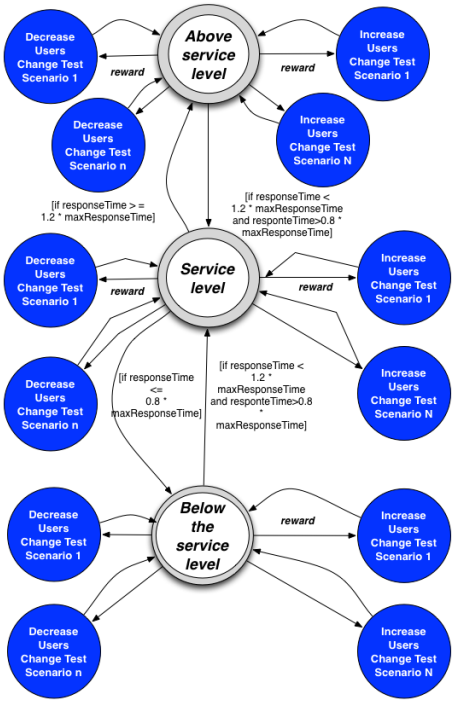


Fig. 13: Hybrid Metaheuristic with Q-Learning Approach



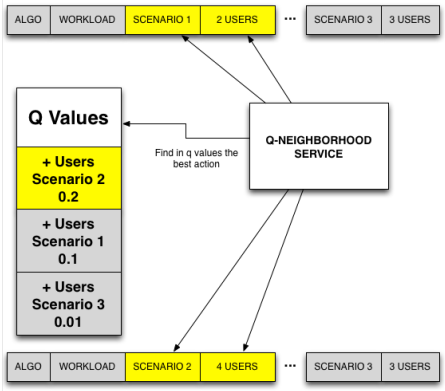


Fig. 15: HybridQ NeighborHood Service

Fig. 14: Markov Decision Process used by HybridQ

Unlike the traditional approach, The update of Q values for each action also occurs in the explotation phase. The exploit phase ends when no value of Q it is 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. The table presents an hyphotetical Q-values for a test. In the table 2, it can be observed

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

that the agents in the Service Level state are in the exploitation phase because there is no other value of Q equals to zero.

The Fig. 15 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.

6.3 IAdapter

IAdapter is a JMeter plugin designed to perform search-based stress tests. The plugin is available on www.iadapter.org. The IAdapter plugin implements the solution proposed in Section 5. The next subsections present details about the Apache JMeter tool, the IAdapter Life Cycle and the IAdapter Components. The IAdapter plugin provides three main components: WorkLoadThreadGroup, WorkLoadSaver, and WorkLoadController. The Fig. 16 show the IAdapter architecture. All metaheuristic class implements the interface IAlgorithm. Test scenarios and test results are stored in a Mysql database. GeneticAlgorithm class uses a framework named JGAP to implement Genetic Algorithms.

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

6.3.1 IAdapter Life Cycle

Fig. 17 presents the IAdapter Life Cycle. The main difference between IAdapter and JMeter tool is that the IAdapter provide an automated test execution where the new test scenarios are choosen by the test tool. In a test with JMeter, the tests scenarios are usually chosen by a test designer.

6.3.2 IAdapter Components

WorkLoadThreadGroup is a component that creates an initial population and configures the algorithms used in IAdapter. Fig. 18 presents the main screen of the WorkLoadThreadGroup component. The component has a name ❶, a set of configuration

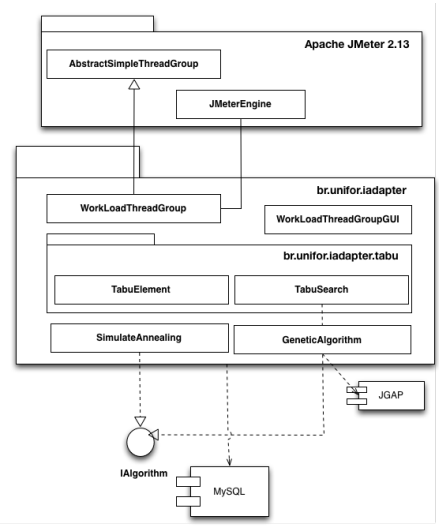


Fig. 16: IAdapter architecture

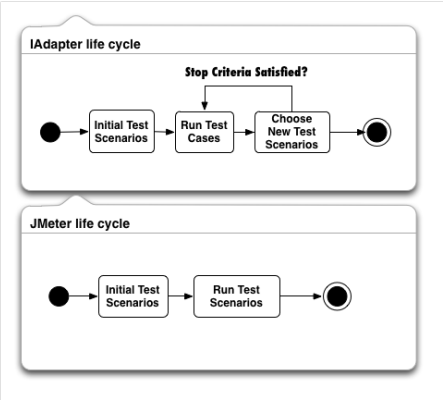


Fig. 17: IAdapter life cycle

tabs ❷, a list of individuals by generation ❸, a button to generate an initial population ❹, and a button to export the results ❺.

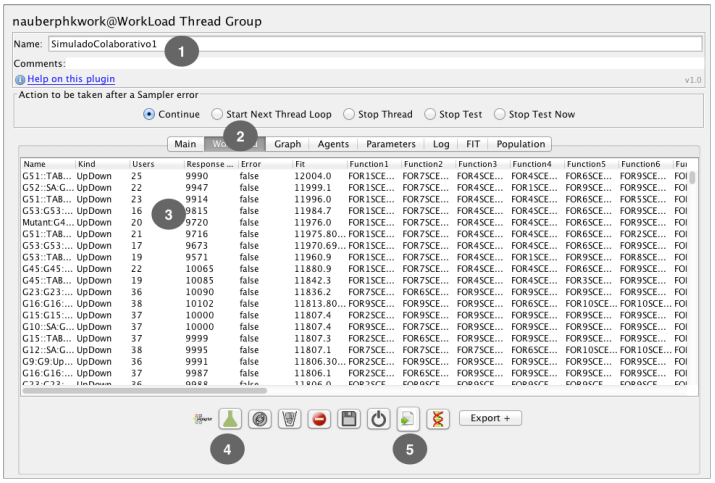


Fig. 18: WorkLoadThreadGroup component

WorkLoadThreadGroup component uses the GeneticAlgorithm, TabuSearch and SimulateAnnealing classes. The WorkLoadSaver component is responsible for saving all data in the database. The operation of the component only requires its inclusion in the test script.

WorkLoadController represents a scenario of the test. All actions necessary to test an application should be included in this component. All instances of the component need to login into the application under test and bring the application back to its original state.

7 Experiments

We conducted two experiments in order to verify the effectiveness of the HybridQ. The first experiment ran for 17 generations in an emulated evinronment. The experiments used an initial population of 4 individuals by metaheuristic. The genetic algorithm used the top 10 individuals from each generation in the crossover operation. The Tabu list was configured with the size of 10 individuals and expired every 2 generations. The mutation operation was applied to 10% of the population on each generation. The experiments uses tabu search, genetic algorithms, the hybrid metaheuristic approach proposed by Gois et al. [10] and the HybridQ approach.

The objective function applied is intended to maximize the number of users and minimize the response time of the scenarios being tested. In this experiments, better fitness values meaning to find scenarios with more users and a low 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. :

$$\begin{aligned} fitness = 3000 * numberOfUsers \\ -20 * 90percentiletime \\ -20 * 80percentiletime \\ -20 * 70percentiletime \\ -20 * maxResponseTime \\ -penalty \end{aligned} \tag{4}$$

The experiments addresses:

- Validate the use of HybridQ algorithm.
- Find the maximum number of users and the minimal response time.
- Analyze and verify the best heuristics among those chosen to the experiments.

All tests in the experiment were conducted without the need of a tester, automating the process of executing and designing performance test scenarios. This experiment applied four scenarios: Two scenarios with peformance problems (Ramp and Circuitous Treasure), scenarios with no performance problems (Happy Scenario 1, Happy Scenario 2) and mixed scenarios.

7.1 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

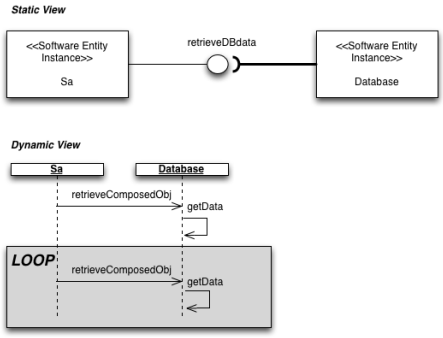


Fig. 19: Circuitous Treasure Hunt sample [50]

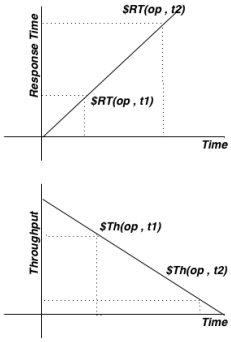


Fig. 20: The Ramp sample [50].

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. 19) [51] [52]. The Ramp it is a 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. 20).

The Fig. 21 presents the fitness value obtained by each metaheuristic. HybridQ metaheuristic obtained the better fitness values.

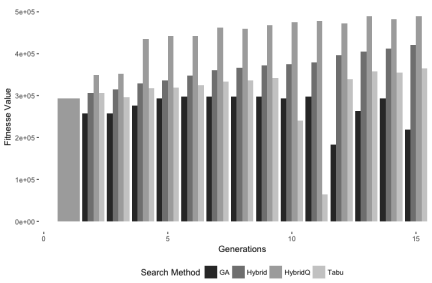


Fig. 21: fitness value obtained by Search Method

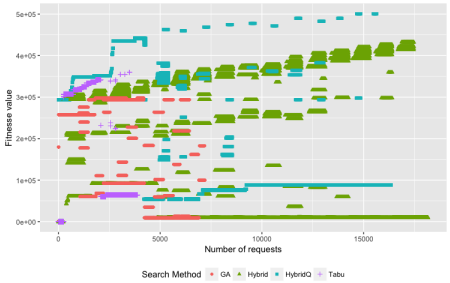


Fig. 22: Number of requests by Search Method

Despite having obtained the best fitness value in each generation, the Hybrid algorithm performs twice as many requests as the tabu search (Fig. 22). The HybridQ algorithm obtained the best fitness value The Fig. 23 shows the average, minimal e maximum value by search method.

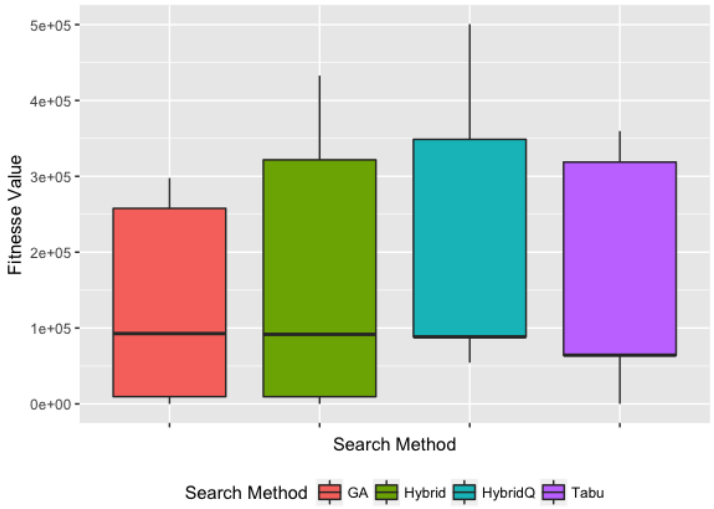


Fig. 23: Average, median, maximum and minimal fitness value by Search Method

The Fig. 24 presents the maximum, average, median and minimum fitness value by generation. The maximun fitness value increases at each generation. The Fig. 25 presents the density graph of number of users by fitness value. The range between 100 and 150 users has the highest number of individuals found with higher fitness value.

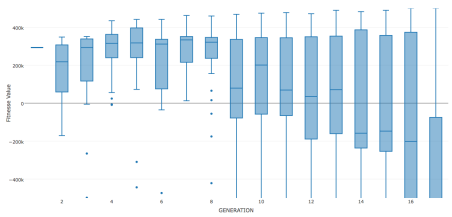


Fig. 24: fitness value by generation

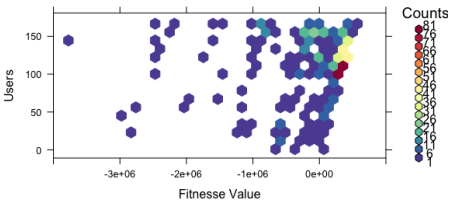


Fig. 25: Density graph of number of users by fitness value

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 on Happy Scenario 2, 16 users on Happy Scenario 1 and a response time of 13 seconds. None of the best individuals has 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. 26 presents the response time by number of users of individuals with Happy Scenario 1 and Happy Scenario 2. The Figure illustrates that the individuals with best fitness value has more users and minor response time.

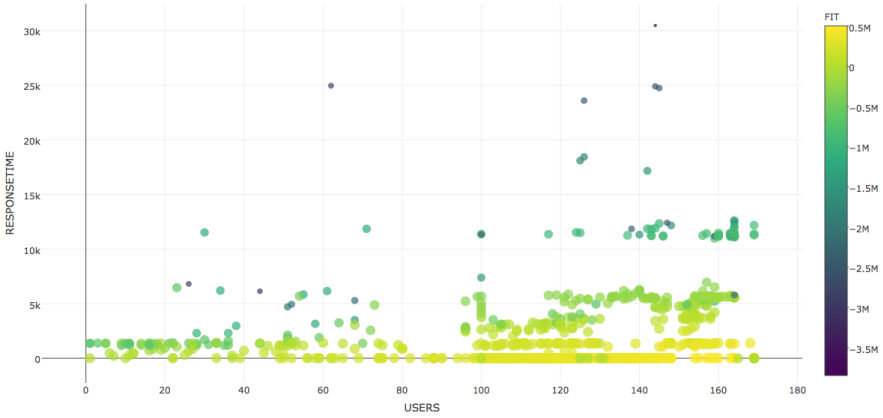


Fig. 26: Response time by number of users of individuals with Happy Scenario 1 and Happy Scenario 2

Fig. 27 presents the markov decision process 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 above the service level, the action with major reward value it is decrease the number of users and include more positions with the Happy Scenario 2. The actions with minor value of reward contais the both antipatterns Circuitous Treasure (CTH) and The Ramp antipatterns (Ramp).

In the first experiment, We conclude that the metaheuristics converged to scenarios with an 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 than Tabu Search to overcome it.

7.2 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:`

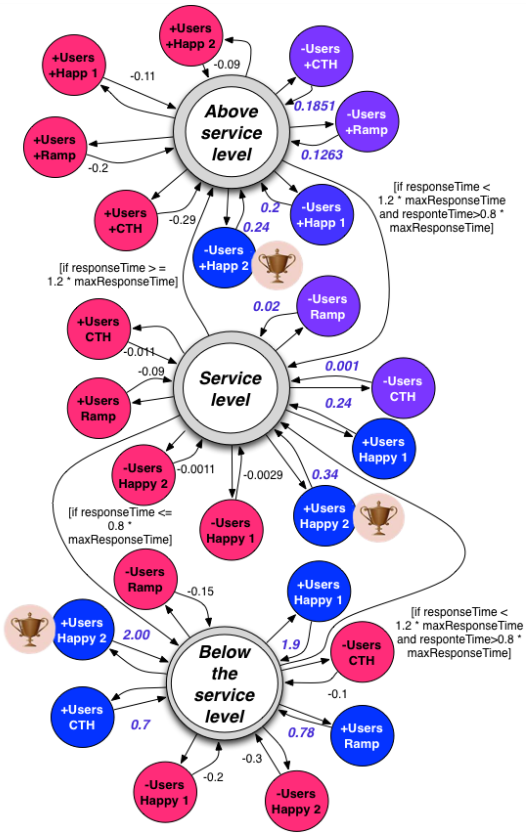


Fig. 27: Markov decision process of experiment with Circuitous Treasure and The Ramp antipatterns

//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 took 2 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 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 realize the register in the system.
- Dogs: The application shows the recorded dogs items.
- Shopping Cart: the application presents the shopping cart.
- Add or Remove in Shopping Cart: the application adds and removes items from shopping cart.

The 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. 28 and 29 presents the search space of individuals found by generation.

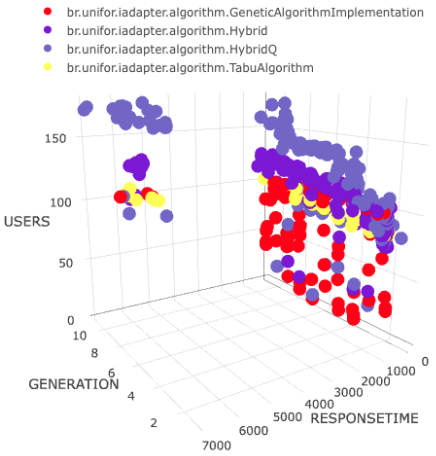


Fig. 28: Search space of individuals found in the experiment by search method

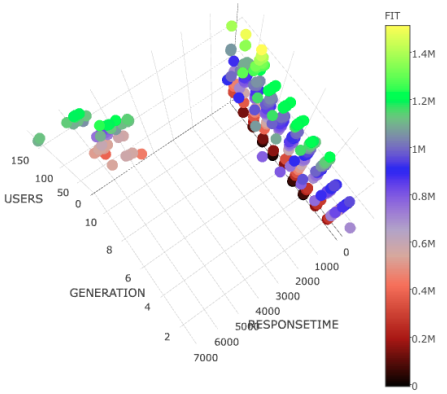


Fig. 29: Search space of individuals found in the experiment by fitness value

The experiment used the following fitness function:

$$fitness = \begin{cases} n * \{3000 * numberOfUsers - 20 * 90 percentiletime - 20 * 80 percentiletime \\ -20 * 70 percentiletime - 20 * maxResponseTime - penalty\}, \text{where n is the} \\ \text{number of features used by the test in a set of previous selected application} \\ \text{features} \end{cases} \tag{5}$$

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 double the fitness value for tests that have the fish and user registration scenario.

This experiment tries to find the scenarios with maximal number of users and best response time tests that contains the Cart and Register features. The Fig. 30 and 31 show the fitness value by generation. The HybridQ obtained the best fitness values in all generations.

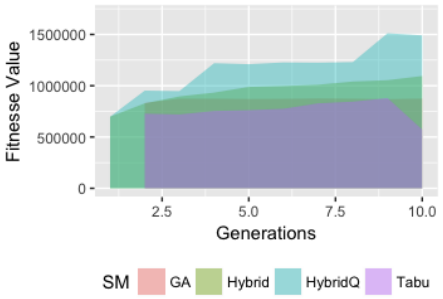


Fig. 30: Fitness value by generation on JPetStore First experiment

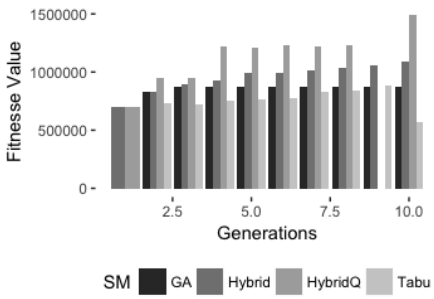


Fig. 31: Fitness value by generation on JPetStore First experiment

The Fig. 32 shows the fitness value by number of request by each Search Method. In the Figure, it is possible to observe that HybridQ obtained the best fitness value to same number of requests of the other algorithms.

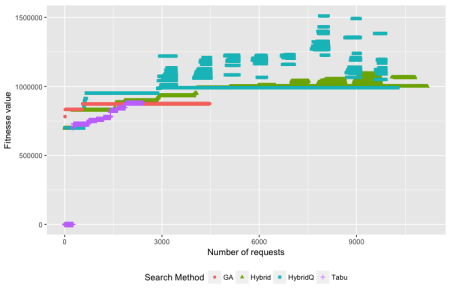


Fig. 32: Number of requests by Search Method

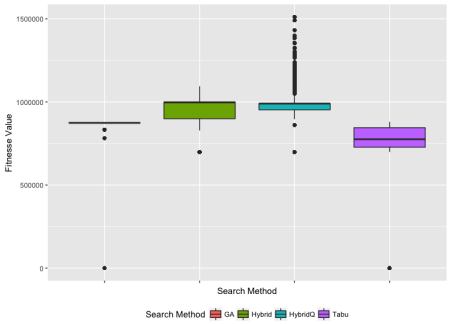


Fig. 33: Fitness value by generation in all tests

Table 5 shows 4 individuals with 233 to 398 users. The first individual has 73 users on fish scenario, 17 users on Dogs scenario, 50 users on Cart scenario, 33 Users on 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 major number of uses. The scenario with major number of users is the Fish search feature. The hybrid meta-

heuristic with Q-Learning (HybridQ) returned individuals with higher fitness scores. The individual with best fitness value has 73 users on fish scenario, 17 users on Dogs scenario, 50 users on Cart scenario, 33 Users on register scenario and a response time of 357 seconds.

8 Conclusion

Two experiments were conducted to validate the proposed approach. The experiments uses genetic, algorithms, tabu search, simulated annealing and an hybrid approach proposed by Gois et al. [10].

The experiments ran for 17 generations. The experiments used an initial population of 4 individuals by metaheuristic. 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 an happy path, excluding the scenarios with the use of an antipatterns. The individual with best fitness value has 64 users on Happy Scenario 2, 81 users on Happy Scenario 1 and a response time of 12 seconds. None of the best individuals has one of the antipatterns used in the experiment.

In the second experiment, HybridQ found the individuals with major number of uses. The scenario with major 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 on fish scenario, 17 users on Dogs scenario, 50 users on Cart scenario, 33 Users on 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 represents global optimum. More experimentation is also required to determine the most appropriate and robust parameters. Lastly, there is a need for an adequate termination criterion to stop the search process.

Among the future works of the research, the use of new combinatorial optimization algorithms such as multi-objective heuristics is one that we can highlight.

References

1. Z. Jiang, Automated analysis of load testing results. Ph.D. thesis (2010). URL `http://dl.acm.org/citation.cfm?id=1831726`
2. I. Molyneaux, *The Art of Application Performance Testing: Help for Programmers and Quality Assurance*, 1st edn. ("O'Reilly Media, Inc.", 2009)
3. D. Draheim, J. Grundy, J. Hosking, C. Lutteroth, G. Weber, in *Conference on Software Maintenance and Reengineering (CSMR'06)* (2006). DOI 10.1109/CSMR.2006.43
4. V. Garousi, IEEE Transactions on Software Engineering **36**(6), 778 (2010). DOI 10.1109/TSE.2010.5
5. C. Babbar, N. Bajpai, D. Sarmah, International Journal of Technology (2011)

6. W.E. Lewis, D. Dobbs, G. Veerapillai, *Software testing and continuous quality improvement* (2005). URL http://books.google.com/books?id=fgaBDd0TfT8C{\&}pgis=1
7. M. Grechanik, C. Fu, Q. Xie, 2012 34th International Conference on Software Engineering (ICSE) pp. 156–166 (2012). DOI 10.1109/ICSE.2012.6227197
8. W. Afzal, R. Torkar, R. Feldt, Information and Software Technology **51**(6), 957 (2009). DOI 10.1016/j.infsof.2008.12.005
9. M.O. Sullivan, S. Vössner, J. Wegener, D.b. Ag, pp. 1–20
10. N. Gois, P. Porfirio, A. Coelho, T. Barbosa, in *Proceedings of the 2016 Latin American Computing Conference (CLEI)* (2016), pp. 718–728
11. M. Harman, P. McMinn, IEEE Transactions on Software Engineering **36**(2), 226 (2010). DOI 10.1109/TSE.2009.71
12. E. Alba, F. Chicano, Computers and Operations Research **35**(10), 3161 (2008). DOI 10.1016/j.cor.2007.01.016
13. A.I. Baars, K. Lakhotia, T.E.J. Vos, J. Wegener, Federated Conference on Computer Science and Information Systems (FedCSIS 2011) pp. 917–923 (2011). URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp={\&}arnumber=6078178
14. P. McMinn, R. Court, S. Testing, P. Street, Software testing, Verification and reliability **14**, 1 (2004). DOI 10.1002/stvr.294
15. V. Garousi, (August) (2006)
16. S. Di Alesio, S. Nejati, L. Briand, A. Gotlieb, IEEE Xplore pp. 158–167 (2013). DOI 10.1109/ISSRE.2013.6698915
17. S. Di Alesio, S. Nejati, L. Briand, A. Gotlieb, Principles and Practice of Constraint Programming pp. 813–830. DOI 10.1007/978-3-319-10428-7{_}58
18. S.D.I. Alesio, L.C. Briand, S. Nejati, A. Gotlieb, ACM Transactions on Software Engineering and Methodology **25**(1) (2015)
19. N.J. Tracey, A search-based automated test-data generation framework for safety-critical software. Ph.D. thesis, Citeseer (2000)
20. J.T.J. Alander, T. Mantere, P. Turunen, in *Neural Nets and Genetic Algorithms* (1998)
21. J. Wegener, H. Sthamer, B.F. Jones, D.E. Eyres, Software Quality Journal **6**(2), 127 (1997). DOI 10.1023/A:1018551716639. URL http://www.springerlink.com/index/uh26067rt3516765.pdf
22. B.J. J. Wegener, K. Grimm, M. Grochtmann, H. Sthamer, EuroSTAR'96: Proceedings of the Fourth International Conference on Software Testing Analysis and Review (1996)
23. L.C. Briand, Y. Labiche, M. Shousha, Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05 p. 1021 (2005). DOI 10.1145/1068009.1068183
24. G. Canfora, M.D. Penta, R. Esposito, M.L. Villani,
25. J. Wegener, M. Grochtmann, Real-Time Systems **15**(3), 275 (1998). DOI Doi10.1023/A:1008096431840
26. F. Mueller, J. Wegener, Proceedings. Fourth IEEE Real-Time Technology and Applications Symposium (Cat. No.98TB100245) (1998). DOI 10.1109/RTTAS.1998.683198
27. P. Puschner, R. Nossal, Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279) (1998). DOI 10.1109/REAL.1998.739738
28. H. Wegener, Joachim and Pitschinetz, Roman and Sthamer, Proceedings of the 1st International Workshop on Automated Program Analysis, Testing and Verification (WAPATV'00) (2000)
29. H. Gross, B.F. Jones, D.E. Eyres, Software, IEE Proceedings- **147**(2), 25 (2000). DOI 10.1049/ip-sen
30. M.D. Penta, G. Canfora, G. Esposito, in *Proceedings of the 9th annual conference on Genetic and evolutionary computation* (2007), pp. 1090–1097
31. V. Garousi, Proceedings of the 10th annual conference on Genetic and evolutionary computation - GECCO '08 p. 1743 (2008). DOI 10.1145/1389095.1389433
32. N.J. Tracey, J.a. Clark, K.C. Mander, (1998)
33. H. Pohlheim, M. Conrad, A. Griep, Analysis (724), 804 (2005). DOI 10.4271/2005-01-0750
34. M. Shousha, Performance stress testing of real-time systems using genetic algorithms. Ph.D. thesis, Carleton University Ottawa (2003)
35. G.R. Raidl, J. Puchinger, C. Blum, in *Handbook of metaheuristics* (Springer, 2010), pp. 469–496
36. C. Blum, A. Roli, ACM Computing Surveys **35**(3), 189 (2003). DOI 10.1007/s10479-005-3971-7
37. E.G. Talbi, *Metaheuristics: From Design to Implementation*, vol. 53 (2013). DOI 10.1017/CBO9781107415324.004
38. W. Jaziri, *Local Search Techniques: Focus on Tabu Search* (2008)

39. F. Glover, R. Martí, Tabu Search pp. 1–16 (1986)
40. C. Raidl, Gunther R and Puchinger, Jakob and Blum, *Hybrid Metaheuristics An Emerging Approach*, vol. 53 (2013). DOI 10.1017/CBO9781107415324.004
41. E.G. Talbi, *Metaheuristics: from design to implementation*, vol. 74 (John Wiley & Sons, 2009)
42. T.P. Hong, H.S. Wang, W.C. Chen, Journal of heuristics **6**(4), 439 (2000)
43. E.G. Talbi, *Hybrid Metaheuristics*, vol. 2 (2012). DOI 10.1007/978-3-642-30671-6
44. R. Raidl, Hybrid Metaheuristics (LNCS 4030) pp. 1–12 (2006). DOI 10.1007/11890584{_}1
45. J. Puchinger, R. Raidl, Artificial Intelligence and Knowledge Engineering Applications a Bioinspired Approach **3562**, 41 (2005). DOI 10.1007/11499305_5
46. R.S. Sutton, A.G. Barto, Learning **3**(9), 322 (2012). DOI 10.1109/MED.2013.6608833. URL https://books.google.com/books?id=CAFR6IBF4xYC{\&}pgis=1$\backslash$nhttp://incompleteideas.net/sutton/book/the-book.html$\backslash$nhttps://www.dropbox.com/s/f4tnuhipchpkgoj/book2012.pdf
47. C. Szepesvári, G. Bartok, Synthesis Lectures on Artificial Intelligence and Machine Learning **4**(x), 1 (2010). DOI 10.2200/S00268ED1V01Y201005AIM009
48. A. Greenwald, K. Hall, R. Serrano, Icml (3), 84 (2003). URL http://www.aaai.org/Papers/Symposia/Spring/2002/SS-02-02/SS02-02-012.pdf
49. C. Blum, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **7505 LNCS**(6), 1 (2012). DOI 10.1007/978-3-642-33860-1_1
50. V. Vetoio, Language (2011)
51. C. Smith, L. Williams, Cmg-Conference- **2**, 797 (2002). URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.6968{\&}rep=rep1{\&}type=pdf
52. C.U. Smith, L.G. Williams, Computer Measurement Group Conference pp. 717–725 (2003). URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.4517{\&}rep=rep1{\&}type=pdf