FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR
Vice-Reitoria de Pesquisa e Pós-Graduação – VRPPG
Programa de Pós-Graduação em Informática Aplicada – PPGIA

Francisco Nauber Bernardo Gois

Search-based Stress Test: an approach appling evolutionary algorithms and trajectory methods

Fortaleza – CE

September / 2017

FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR
Vice-Reitoria de Pesquisa e Pós-Graduação – VRPPG
Programa de Pós-Graduação em Informática Aplicada – PPGIA

Francisco Nauber Bernardo Gois

Search-based Stress Test: an approach appling evolutionary algorithms and trajectory methods

A thesis submitted to the Department of Applied Informatics (PPGIA) of Universidade de Fortaleza (UNIFOR), in partial fulfillment of the requirements for the degree of Doctor os Science in Applied Informatics.

Thesis advisor: Prof. Dr. Pedro Porfírio Muniz de Farias
Thesis co-advisor: Prof. Dr. André Luís Vasconcelos Coelho

Fortaleza – CE

September / 2017

Francisco Nauber Bernardo Gois

Search-based Stress Test: an approach appling evolutionary algorithms and trajectory methods

Data de Aprovação: __/____

Banca Examinadora:

D.Sc. Pedro Porfírio Muniz de Farias (UNIFOR)

D.Sc. André Luís Vasconcelos Coelho (UNIFOR)

D. Sc. Pedro de Alcantara dos Santos Neto (UFPI)

D. Sc. João Paulo Pordeus Gomes (UFC)

Ph.D. Americo Tadeu Falcone Sampaio (UNIFOR)

Docteur Napoleão Vieira Nepomuceno (UNIFOR)

Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning. (Albert Einstein)

ACKNOWLEDGMENTS

It has been an exciting and instructive study period at the University of Fortaleza and I feel privileged to have had the opportunity to carry out this study as a demonstration of knowledge gained during the period. With these acknowledgments, it would be impossible to remember those who in one way or another, directly or indirectly, have played a role in the realization of this research project.

First of all, Thank you God for the blessing and guiding me during all this project. I would like to dedicate this thesis to three of the most important people in my life – my wife, my mother and my father – whose unconditional love and support gave me the strength to follow my dream of becoming a scientist. I express all my gratitude to my beloved wife Najara, who endured five long years by my side without ever questioning my choice of becoming a doctoral degree student. Thank you to my parents, Gois and Zirlanda, for the example of life and for encouraging me as a child to enjoy science.

I would like to thank my very dedicated advisors Dr. Pedro Porfírio and André Coelho, for their guidance and support throughout this research project, which has been an excellent learning experience for me. This thesis is not solely the product of my work but also the hard work and perseverance of my advisors, providing help to keep me going even in the most difficult of times to get the project done. I can only hope that more students will be given the opportunity to work and learn from both teachers as I did.

I also would like to thank the following people:

My thesis committee – Dr. Pedro de Alcantara dos Santos Neto, Dr. João Paulo Pordeus Gomes, Ph.D. Americo Tadeu Falcone Sampaio and Docteur Napoleão Vieira Nepomuceno – who made themselves available when I needed them to review previous drafts of this thesis and for their valuable suggestions that helped me to bring this work to its completion.

My family who loves me and is always present for good or bad: my mother in law Aroldina, my brothers Nilton and Nelio, my aunt Zena and my grandmother Elisabete.

Adriano Bessa, Victor Hugo and all professors from the post-graduate program for their direct and indirect contributions.

My colleagues from UNIFOR who supported me during this project: Adriana, Marum Simão, Celso Medeiros and José Renato.

My company, SERPRO—Brazilian Federal Datacenter, for the (partial) financial support. My colleagues Flavio Cysne, Fred Viana and Suderland Guimarães to help me with one of the experiments applied in this thesis. Sergio Gomes and Carlos Henrique (Managers of the Fortaleza Team) for giving me the time necessary to develop my activities.

Julio Tavares, Carlos Manso and Marcia Tonieto. Coordinators of the Computer Science Course at Faculdade Metropolitana da Grande Fortaleza - Fametro and Faculdade Lourenço Filho. For all your support over the years that I taught in those colleges.

RESUMO

Alguns sistemas de software devem responder a milhares ou milhões de pedidos simultâneos. Esses sistemas devem ser devidamente testados para garantir que eles possam funcionar corretamente sob a carga esperada. A degradação do desempenho e conseqüentes falhas do sistema geralmente ocorrem em condições de estresse. O teste de estresse sujeita o programa a cargas pesadas. Os testes de estresse diferem de outros tipos de testes em que o sistema é executado em seus pontos de interrupção, forçando o aplicativo ou a infra-estrutura de suporte a falhar. A busca do tempo de execução mais longo é vista como um problema de otimização descontínuo, não-linear, com o domínio de entrada do sistema em teste como espaço de busca. Neste contexto, os testes baseados em pesquisa (*search-based tests*) são vistos como uma abordagem promissora para verificar as restrições de tempo. O teste de software baseado em pesquisa é a aplicação de técnicas de pesquisa metaheurística para gerar testes de software. O critério de adequação do teste é transformado em uma função de fitness e um conjunto de soluções no espaço de busca é avaliado em relação à função de fitness usando uma metaheurística. O teste de estresse baseado em pesquisa envolve encontrar os tempos de execução melhores e piores para verificar se as restrições de tempo são cumpridas. Os acordos de nível de serviço (SLA) são documentos que especificam garantias de desempenho realistas, bem como penalidades por incumprimento. Os SLAs são feitos entre provedores e clientes que incluem qualidade do serviço, capacidade de recursos, escalabilidade, obrigações e conseqüências em caso de violação. Satisfazer o SLA é de grande importância e um problema desafiador. A principal motivação desta tese é encontrar o tempo de resposta adequado dos SLAs usando teste de estresse. Esta tese aborda três abordagens em testes de estresse baseados em pesquisa. Primeiro, a metaheurística híbrida usa Tabu Search, Simulated Annealing e Algoritmos Genéticos de forma colaborativa. Em segundo lugar, uma abordagem chamada HybridQ usa uma técnica de aprendizado de reforço para otimizar a escolha de soluções vizinhas para explorar, reduzindo o tempo necessário para obter os cenários com o tempo de resposta mais longo na aplicação. As melhores soluções encontradas pelo HybridQ foram em média 5,98 % melhores que alcançados pela abordagem híbrida sem Q-learning. Em terceiro lugar, a tese investiga o uso dos algoritmos multi-objetivos NSGA-II, SPEA2, PAES e MOEA/D. A metaheurística MOEA/D obteve o melhor valor de hipervolume quando comparada com outras abordagens. A abordagem colaborativa usando MOEA/D e HybridQ melhora os valores de hipervolume obtidos e encontrou *workloads* mais relevantes do que as experiências anteriores. Uma ferramenta chamada IAdapter, um plugin JMeter para realizar testes de esforço baseados em pesquisa, foi desenvolvida e usada para realizar todas as experiências.

Palavras-chave: Search-based Testing, Stress Testing, Multi-objective metaheuristics, Hybrid metaheuristics, Reinforcement Learning.

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. Stress tests differ from other kinds of testing in that the system is executed on its breakpoints, forcing the application or the supporting infrastructure to fail. The search for the longest execution time is seen as a discontinuous, nonlinear, optimization problem, with the input domain of the system under test as a search space. In this context, search-based testing is viewed as a promising approach to verify timing constraints. Search-based software testing is the application of metaheuristic search techniques to generate software tests. The test adequacy criterion is transformed into a fitness function and a set of solutions in the search space is evaluated with respect to the fitness function using a metaheuristic. Search-based stress testing involves finding the best- and worst-case execution times to ascertain whether timing constraints are fulfilled. Service Level Agreements (SLAs) are documents that specify realistic performance guarantees as well as penalties for non-compliance. SLAs are made between providers and customers that include service quality, resources capability, scalability, obligations, and consequences in case of violations. Satisfying SLA is of great importance and a challenging issue. The main motivation of this thesis is to find the adequate response time of SLAs using Stress Testing. This thesis addresses three approaches in search-based stress tests. First, Hybrid metaheuristic uses Tabu Search, Simulated Annealing, and Genetic Algorithms in a collaborative manner. Second, an approach called HybridQ uses 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 best solutions found by HybridQ were on average 5.98% better that achieved by the Hybrid approach without Q-learning. Third, the thesis investigates the use of the multi-objective NSGA-II, SPEA2, PAES and MOEA/D algorithms. MOEA/D metaheuristics obtained the best hypervolume value when compared with other approaches. The collaborative approach using MOEA/D and HybridQ improves the hypervolume values obtained and found more relevant workloads than the previous experiments. A tool named IAdapter, a JMeter plugin for performing search-based stress tests, was developed and used to conduct all the experiments.

Keywords: Search-based Testing, Stress Testing, Multi-objective metaheuristics, Hybrid metaheuristics, Reinforcement Learning.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

BCET - Best-Case Execution Time .

CP - Constraint Programming.

GA - Genetic Algorithms.

EMO - Evolutive Multi-objective algorithm.

FOREPOST - Feedback-ORiEnted PerfOrmance Software Testing.

Hybrid -Hybrid Metaheuristic approach proposed by this research.

HybridQ - Hybrid Metaheuristic with Q-Learning approach proposed by this r.

ICA - Independent Component Analysis.

MBT - Model-based Testing.

MDP - Markov Decision Process.

MOEA/D - Multiobjective Evolutionary Algorithm Based on Decomposition.

MOEAs - Multi-objective evolutionary algorithms.

MOPs - Multi-objective problems.

NIST - National Institute of Standards and Technology.

NSGA II - Non-dominated Sorting Genetic Algorithm II.

P-metaheuristics - Population-based metaheuristics.

PAES - Pareto Archived Evolution Strategy.

PASA - A Method for Performance Assessment of Software Architectures.

PID - Proportional,Integral, and Derivative.

S-metaheuristic - Single Solution or Single-objective metaheuristic.

SA - Simulated Annealing.

SBSE - Search-based Software Engineering.

SBST - Search-based Software Testing.

SEDR - Standard Error Dynamic Resampling.

SLAs - Service Level Agreements.

SPEA2 - Strength Pareto Evolutionary Algorithm 2.

TPC - Transaction Processing Performance.

TS - Tabu Search.

UCML - User Community Modeling Language.

WCET - Worst-Case Execution Time .

GLOSSARY OF TERMS

EMOA package - R package design to analysis of evolutionary multiobjective optimization algorithms.

IRace - Algorithm procedure to automatize the configuring parameters of metaheuristics (Manuel López-Ibãnez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, 2016).

Individual - In the context of evolutionary algorithms, individuals represent each solution found or used by the evolutionary algorithm (Gendreau, Michel and Potvin, 2010).

GUI objects - Graphical user interface elements are those elements used by graphical user interfaces to offer a consistent visual language.

Offspring - New solutions created to new to a new generation in evolutionary algorithms(Gendreau, Michel and Potvin, 2010).

Outliers - An observation that is well outside of the expected range of values in a study or experiment, and which is often discarded from the data set (Gendreau, Michel and Potvin, 2010).

Pareto optimal- Set of non-dominated solutions in a search-based problem (DEB, 2001).

WorkLoad - Represents the size of the demand that will be imposed on the application under test in an execution. In this thesis, each workload represent an individual in the search space.

RELATED PUBLICATIONS

The following publications are related to this thesis:

N. Gois, P. Porfirio and A. Coelho. A multi-objective metaheuristic approach to search-based stress testing. In Proceedings of the 2017 IEEE International Conference on Computer and Information Technology (CIT), 2017 (GOIS; PORFIRIO; COELHO, 2017).

N. Gois, P. Porfirio, A. Coelho, and T. Barbosa. Improving Stress Search Based Testing using a Hybrid Metaheuristic Approach. In Proceedings of the 2016 Latin American Computing Conference (CLEI), pages 718–728, 2016 (GOIS et al., 2016).

The following publications are submited and under review:

N. Gois, P. Porfirio and A. Coelho.Improving Search-Based Stress Testing Using Q-Learning and a Hybrid Metaheuristic Approach. Journal of Software Engineering Research and Development.

1 INTRODUCTION

1.1 Motivation

Performance problems such as high response times in software applications have a significant effect on the customers' satisfaction. The explosive growth of the Internet has been instrumental in the increased need of applications that perform at an appropriate speed. Moreover, 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. 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 (JIANG, 2010) (MOLYNEAUX, 2009) (WERT et al., 2014).

Software testing is an expensive and difficult activity. The exponential growth in the complexity of software makes the cost of testing continue to grow. Test case generation can be viewed as a search problem. The test adequacy criterion is transformed into a fitness function and a set of solutions in the search space are evaluated with respect to the fitness function using a metaheuristic search technique. Search-based software testing is the application of metaheuristic search techniques to generate software tests cases or perform test executions (AFZAL; TORKAR; FELDT, 2009).

There is strong empirical evidence, that deficient testing of both functional and non-functional properties is one of the major sources of software and system errors. In 2002, NIST report found that more than one-third of these costs of software failure could be eliminated by an improved testing infrastructure. Automation of testing is a key concern. Through automation, large-scale thorough testing can become practical and scalable. However, automated generation of test cases presents challenges. The general problem involves finding a (partial) solution to the path sensitization problem. That is the problem of finding an input to drive the software down a chosen path (HARMAN; MCMINN, 2010) (DEAN; DON, 2003).

Software performance is a pervasive quality, because it is influenced by every aspect of the design, code, and execution environment. Performance failures occur when a software product is not able to meet its overall objectives due to inadequate performance. Such failures negatively impact the projects by increasing costs, decreasing revenue or both (TRUBIANI, 2011b). Stress testing of enterprise applications is manual, laborious, costly, and not particularly effective. When running many different test cases and observing application's behavior, testers intuitively sense that there are certain properties of test cases that are likely to reveal performance bugs (GRECHANIK; FU; XIE, 2012). Manual analysis of load testing is inefficient and error prone due to incomplete knowledge of test analyst about the application under test(ARSLAN et al., 2015).

Typically, stress testing is accomplished using test scripts, which are programs that test designers write to automate testing. These test scripts perform actions or mimicking user actions on GUI objects of the system to feed input data. Contemporary approaches to load testing suffer from limitations. Their cost-effectiveness depends heavily on the actual 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 problem (GRECHANIK; FU; XIE, 2012).

Stress testing is an expensive and difficult activity. The exponential growth in the complexity of software makes the cost of testing has only continued continued to grow. Test case generation can be seen as a search problem. The test adequacy criterion is transformed into a fitness function and a set of solutions in the search space are evaluated with respect to the fitness function using a metaheuristic search technique. Search-based software testing is the application of metaheuristic search techniques to generate software tests cases or perform test execution (AFZAL; TORKAR; FELDT, 2009).

Search-based stress testing (SBST) is regarded as a promising approach to verify timing constraints (AFZAL; TORKAR; FELDT, 2009). A common objective of a load search-based test is to find scenarios that produce execution times that violate the specified timing constraints (SULLIVAN et al., 1998).

Experimentation is essential in order to realistically and accurately evaluate search-based stress tests. Experimentation on algorithms is usually made by simulation. Experiments involving search-based stress tests are inherently complex and typically time-consuming and extremely difficult to repeat. Ordinarily, People who might want to duplicate published results, for example, must devote substantial resources to set up and the environmental conditions are likely to be different.

Usually, search-based test methods are based on a single objective optimization. Multi-objective evolutionary algorithms (MOEAs) are commonly used for solving multi-objective problems (MOPs) because they produce a complete set of solutions in a single run. Consequently, multi-objective heuristics may be more suitable in non-functional search-based tests, since these tests usually aim to obtain a result with more than one objective, for example, maximize the number of users of an application, minimizing your response time.

Most of the commercial or academic tools don't use MOEAs in search-based stress tests. Several tools provide support for processing and running tests with different approaches. Moreover, while some of the tools use model-based tests, others set of tools use a scripting-based approach.Stress testing tools are essential in order to verify compliance with the service levels of an application.

Service Level is the service the customer hopes to receive (PARASURAMAN; BERRY; ZEITHAML, 1991). Service Level Agreements (SLAs) are documents that specify realistic performance guarantees as well as penalties for non-compliance (MENASCÉ, 2002). SLAs are made between providers and customers that include service quality, resources capability, scalability, obligations and consequences in case of violations. Satisfying SLA is very important and a challenging issue (RAJESHWARI, 2016). Decision making and definition of an SLA involve finding the appropriate response times for a test application. The main motivation of this thesis is to propose a new set of techniques that could help to define the response time of SLAs using search-based stress testing. The next sections introduce the Research Questions, Contributions and the Thesis Outline.

1.2 Research Questions

This thesis addresses the use of hybrid and multi-objective metaheuristics in conjunction with reinforcement learning techniques in search-based stress tests. A tool named IAdapter (www.iadapter.org, github.com/naubergois/newiadapter), a JMeter plugin for performing search-based stress tests, was developed. The thesis uses a Noise Reduction algorithm named SEDR in the experiments of the chapter 6. Considering these motivations and the objective of this thesis, the main research questions which this thesis addresses are:

- **(Research Question 1)** How to improve search-based stress testing using single-objective metaheuristics?
- **(Research Question 2)** How to improve the choice of neighboring solutions in a single-

objective metaheuristic to explore, reducing the time needed to obtain the scenarios with the longest response time in the application?

- **(Research Question 3)** How to use multi-objective metaheuristics in search-based stress testing to obtain a Pareto frontier to improve the definition of SLAs?

In the next section, we show the main contributions of this thesis, based on the presented research questions.

1.3 Contributions

The main contributions of this thesis are follows:

- We propose a hybrid algorithm approach in search-based stress testing. The approach uses Tabu Search, Simulated Annealing and Genetic Algorithms. The thesis presents three experiments. The results were published in (GOIS et al., 2016). This proposal refers to our **Research Question 1** (Chapter 4).

- We present a hybrid algorithm with the Q-Learning approach. The approach uses reinforcement learning to optimize the choice of neighboring solutions, reducing the time required to obtain the scenarios with the longest response time. This proposal refers to our **Research Question 2** (Chapter 5).

- We investigate the benefits of multiobjective metaheuristics in search-based stress testing. A comprehensive investigation of the use of multi-objective metaheuristics on search-based stress testing was conducted. Part of the results was published in (GOIS; PORFIRIO; COELHO, 2017).This proposal refers to our **Research Question 3** (Chapter 6).

The secondary contributions of this thesis are follows:

- The Survey of Stress Testing presented in Appendix A.

- A Noise Reduction approach extended of the SEDR algorithm presented in Appendix B.

- The IAdapter tool presented in Appendix C.

1.4 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2 presents initial concepts about the use of metaheuristics, hybrid metaheuristics, multi-objective metaheuristics and reinforcement learning. The chapter briefly introduces the Trajectory and Population-based Metaheuristics. Subsequently, presents Hybrid and Multi-Objective metaheuristics. Finally, presents the Reinforcement Learning concepts.

Chapter 3 shows the context where this thesis is inserted and contemporary concepts about Search-based Stress Testing. In particular, two categories of approaches are out-lined: (i) Search-Based Stress Testing on Safety-critical systems; (ii) Search-Based Stress Testing on non Safety-critical systems.

Chapter 4 presents a hybrid approach that combines genetic algorithms, simulated annealing, and tabu search algorithms in stress tests. A tool named IAdapter, a JMeter plugin for performing search-based load tests, was developed. Three experiments were performed to validate the proposed approach. The first experiment was performed on an emulated component, and the second one was performed using an installed Moodle application. The chapter presents the solution representation and the objective function of the proposed approach.

Chapter 5 presents a reinforcement learning approach 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 chapter presents a study that extends the article "Improving stress search based testing using a hybrid metaheuristic approach" in order to ascertain if the use of the Q-learning technique allows the meta-heuristic algorithms to improve the search to 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.

Chapter 6 presents experiments to assert the benefits of multiobjective metaheuristics in search-based stress testing. The first experiment uses NSGA-II algorithm to discover application scenarios where there is a high response time for a small number of users. The second experiment presents the experimental results to compare four multi-objective algorithms in search-based stress testing. The third experiment shows the results conducted to compare the MOEA/D multi-objective with the HybridQ algorithm. The fourth experiment presents

an experiment to evaluate the collaborative approach using HybridQ and MOEA/D. The fifth experiment presents an initial study to evaluate the Pareto Frontier difference in multiple executions. Finally, we present a survey study to assess the use of multi-objective algorithm by the non-functional test team of the Serpro, a company of the Brazilian Federal Government in Fortaleza.

2 METAHEURISTICS AND REINFORCEMENT LEARNING

This chapter presents concepts and essentials definitions considering the main contributions of this thesis: the single and multi-objective metaheuristics and reinforcement learning.

2.1 Metaheuristics

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 (BLUM; ROLI, 2003):

- 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 (Trajectory methods). 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.

2.1.1 Trajectory Methods

Trajectory methods are characterized by a trajectory in the search space. Two common trajectory methods are Simulated Annealing and Tabu Search.

Neighborhood

The definition of Neighborhood is a required common step in the design of any Single-Solution metaheuristic (S-metaheuristic). The neighborhood structure it is an important piece of 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 (TALBI, 2013).

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 (TALBI, 2013). The Fig. 2.1 presents an example where a set of neighbors is found by permutation.

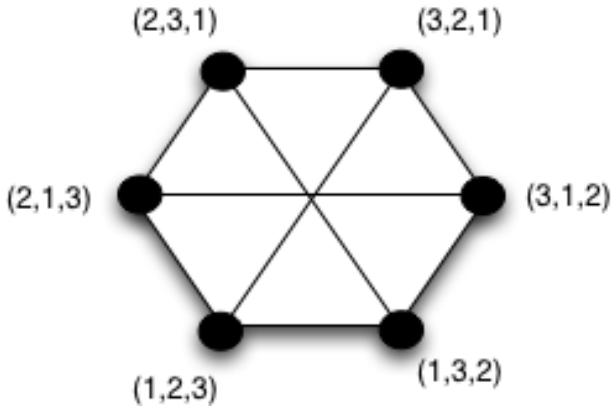


Figure 2.1: An example of neighborhood for a permutation (TALBI, 2013).

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.

Simulated Annealing

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

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
```

Algorithm 2 Tabu Search Algorithm

```
    s ← GenerateInitialSolution()
2:  InitializeTabuLists(TL1,...,TLr)
    while termination conditions not met do
4:      Na(s) ← {s1 ∈ N(s)|s1 does not violate a tabu condition, or it satisfies at least one aspiration condition }
        s1 ← argmin{f(s2)|s2 ∈ Na(s)}
6:      UpdateTabuLists(TL1,...,TLr,s,s1)
        s ← s1
8:  end while
```

PickNeighborAtRandom(N(s)). If s_1 is better than s, then s_1 is accepted as a 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 (RAIDL; PUCHINGER; BLUM, 2010).

Tabu Search

Tabu Search uses a tabu list to keep track of the last moves, and don't allow going back to these (GLOVER; MARTÍ, 1986). 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.

2.1.2 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 population is integrated into the current one using 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 (TALBI, 2009).

Population-based metaheuristics are comprised of several components (HONG; WANG; CHEN, 2000) (SHOUSHA, 2003) :

- a representation of the solution, referred as the chromosome;
- fitness of each chromosome, referred 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 (BLUM; ROLI, 2003). 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 (ALETI; MOSER; GRUNSKE, 2016). 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 (RAIDL; PUCHINGER; BLUM, 2010).

Algorithm 3 Genetic Algorithm

1: $s \leftarrow GenerateInitialSolution()$
2: Evaluate(P)
3: **while** termination conditions not met **do**
4: $\quad P_1 \leftarrow Recombine(P)$
5: $\quad P_2 \leftarrow Mutate(P_1)$
6: $\quad Evaluate(P_2)$
7: $\quad P \leftarrow Select(P_2, P)$
8: **end while**

2.1.3 Hybrid Metaheuristics

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 (RAIDL; PUCHINGER; BLUM, 2010).

There are two main categories of metaheuristic combinations: collaborative combinations and integrative combinations. These are presented in Fig. 2.2 (RAIDL, 2006).

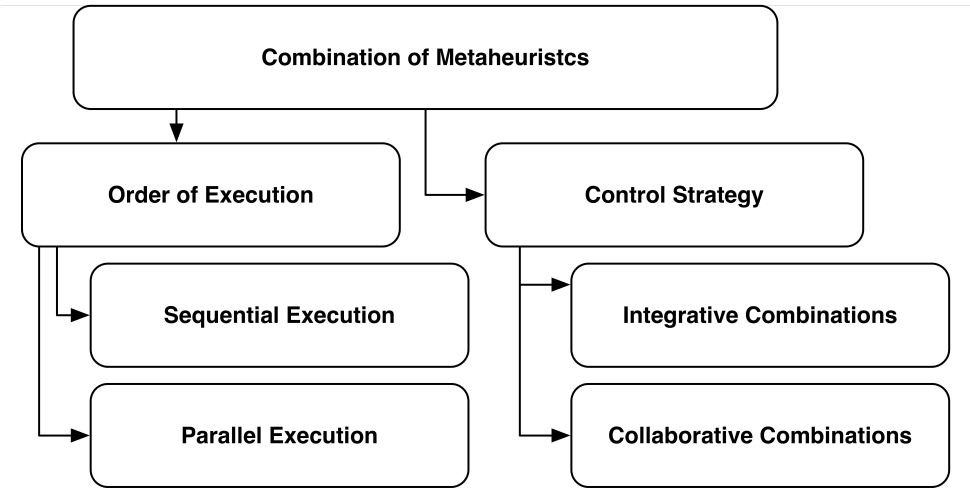


Figure 2.2: Categories of metaheuristic combinations (PUCHINGER; RAIDL, 2005)

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.

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 of 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 (RAIDL; PUCHINGER; BLUM, 2010).

2.1.4 Multi-objective Metaheuristics

Many real optimization problems require optimizing multiple conflicting objectives. There is no single optimal solution, but a set of alternative solutions. The objectives that have to be optimized are often in competition with one another and may be contradictory; we may find ourselves trying to balance the different optimization objectives of several distinct goals (HARMAN; MCMINN, 2010). The image of all the efficient solutions is called the Pareto front or Pareto curve or surface. The shape of the Pareto surface indicates the nature of the trade-off between the different objective functions. An example of a Pareto curve is reported in Fig. 2.3. Multi-objective optimization methods have as main purposes to minimize the distance between the nondominated front and the Pareto optimal front and find a set of solutions that are as diverse as possible.

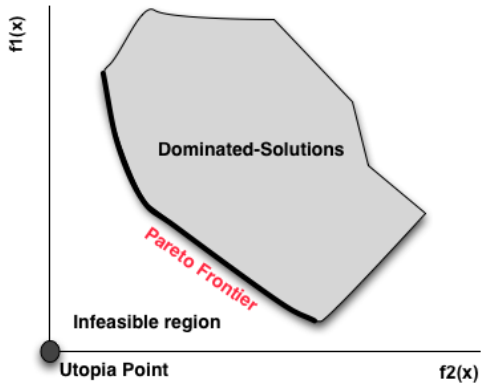


Figure 2.3: An optimized Pareto front example

What distinguishes multi-objective evolutionary algorithms from single objective metaheuristics is how they rank and select individuals in the population. If there is only one objective, individuals are naturally ranked according to this objective, and it is clear which individuals are best and should be selected as parents. In the case of multiple objectives, it is still necessary to rank the individuals, but it is no longer obvious how doing this. Most people probably agree that a good approximation to the Pareto front is characterized by:

- a small distance of the solutions to the true Pareto frontier,
- a wide range of solutions, i.e., an approximation of the extreme values, and
- a good distribution of solutions, i.e., an even spread along the Pareto frontier.

The approximation of the Pareto-optimal set involves itself two objectives: minimize the distance to the optimal front and maximize the diversity of the generated solutions. There

are two fundamental issues when designing a multiobjective evolutionary algorithm: mating selection and environmental selection. The first issue is related to the question of how to guide the search towards the Pareto-optimal front. The procedure to fill the mating pool is usually randomized. The second issue is related to the question of which individuals to keep during the evolution process. In most modern multi-objective algorithms these two concepts are realized in the following way: Environmental selection or Mating selection (ZITZLER; LAUMANNS; THIELE, 2001).

In Environmental selection, an archive is maintained which contains a representation of the nondominated front among all solutions considered so far. A member of the archive is only removed if i) a solution has been found that dominates it or ii) the maximum archive size is exceeded and the portion of the front where the archive member is located is overcrowded (ZITZLER; LAUMANNS; THIELE, 2001).

In Mating selection, the pool of individuals is evaluated in two phases. First, all individuals are compared on the basis of the Pareto dominance. Basically, the information which individuals each individual dominates, is dominated by or is indifferent to is used to define a ranking on the generation pool. Afterwards, this ranking is refined by the incorporation of density information. Various density estimation techniques are used to measure the size of the niche in which a specific individual is located (ZITZLER; LAUMANNS; THIELE, 2001).

NSGA-II Multi-objective Heuristics

Multi-objective metaheuristics rank individuals according to the defined goals. Deb et al. proposed the Nondominated Sorting Genetic Algorithm II (NSGA-II) algorithm taking into account the need to reduce computational complexity in non-dominated classification while introducing elitism and eliminating subjectivity in the allocation of the sharing parameter (DEB et al., 2000). NSGA-II is a multi-objective algorithm, based on GAs, and implements the concept of dominance, in other words, to classify the total population in fronts according to the degree of dominance. According to NSGA-II, the individuals that are located on the first front are considered the best solutions of that generation, while in the last front are the worst. Using this concept, one can find more consistent results, located closer to the Pareto region, and better adapted to the type of problem.

The NSGA algorithm II applies a fitness evaluation in an initial population (Figure 2.4- ❶ and ❷). The populations are ranked using multiple tournament selections, which consist of comparing two solutions (Figure 2.4- ❸). In order to estimate the density of the solutions surrounding a particular solution in the population, the common distance between the previous

solution and the posterior is calculated for each of the objectives. This distance serves as an estimate of the size of the largest cuboid that includes solution i without including any other solution of the population. A solution i beats another solution if:

- Solution i has a better rank, then $Rank_i < Rank_j$.
- Both solutions have the same rank, but i has a greater Distance than j, then $Rank_i = Rank_j$ and $Distance_i > Distance_j$.

At the end of each analysis a certain group of individuals is classified as belonging to a specific category called the front, and upon completion of the classification process, all individuals will be inserted into one of the n fronts. Front 1 is made up of all nondominated solutions. Front 2 can be achieved by considering all nondominated solutions excluding solutions from front 1. For the determination of front 3, solutions previously classified on front 1 and 2 are excluded, and so on until all individuals have been classified on some front.

After selection, recombination and mutation are performed as in conventional GAs (Figure 2.4- ❹). The two sets (father and son of the same dimension) are united in a single population (dimension 2) and the classification is applied in dominance fronts. In this way, elitism is guaranteed preserving the best solutions (fronts are not dominated) in the latest population (Figure 2.4- ❻).

However, not all fronts can be included in the new population. Thus, Deb et al. proposed a method called crowd distance, which combines the fronts not included in the set, to compose of the last spaces of the current population, guaranteeing the diversity of the population (DEB et al., 2000). The NSGA-II algorithm creates a set of front lines, in which each front containing only non–dominating solutions. Within a front, individuals are rewarded for being 'spread out'. The algorithm also ensures that the lowest ranked individual of a front still has a greater fitness value than the highest ranked individual of the next front (YOO; HARMAN, 2007).

SPEA2: Improving the Strength Pareto Evolutionary Algorithm

SPEA uses a regular population and an archive. Starting with an initial population and an empty archive the following steps are performed per iteration. First, all non-dominated population members are copied to the archive; any dominated individuals or duplicates are removed. If the size of the updated archive exceeds a predefined limit, further archive members are deleted by a clustering technique which preserves the characteristics of the non-dominated front. Afterwards, fitness values are assigned to both archive and population members. Each individual i

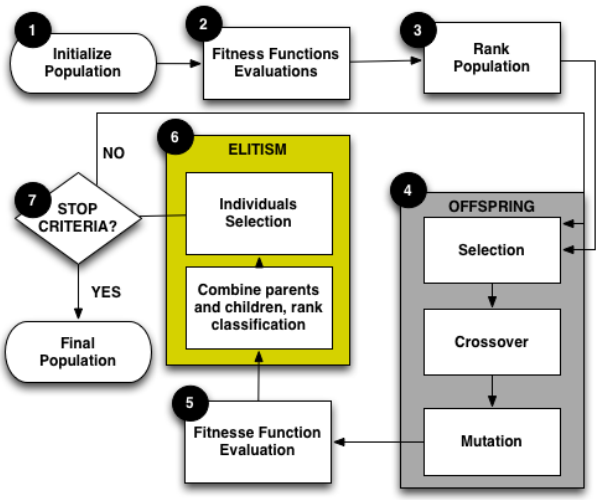


Figure 2.4: NSGA-II Algorithm

in the archive assign a strength value S(i) ∈ [0, 1], which at the same time represents its fitness value F(i). 0 indicates a non-dominated individual,whereas a high value points out that the individual is dominated by many other ones. S(i) is the number of population members j that are dominated by or equal to i with respect to the objective values, divided by the population size plus one. The algorithmic framework of SPEA2 is described in Alg. 4. An initial population P_0 and an initial archive are created. The fitness value of all individuals is calculated in population and in the archive (external set). All non-dominated individual are copied to the new archive. Finally, the algorithm select the individual using a tournament selection (ZITZLER; LAUMANNS; THIELE, 2001) (TERVONEN; KINGDOM, 2017) (Matnei Filho; VERGILIO, 2016).

The main differences between SPEA2 and NSGA-II are the diversity assignment and replacement. NSGA-II uses a fast non-dominated sorting algorithm and uses Pareto optimal levels as the primary criterion to select solutions. SPEA2 derives the strength of each solution from the number of other solutions it dominates. NSGA-II uses the crowding-distance to maintain a well-spread set of solutions whereas SPEA2 applies the k-nearest neighbor approach (Figure 2.5) (TERVONEN; KINGDOM, 2017) (DEB; MOHAN; MISHRA, 2005).

Pareto Archived Evolution Strategy

Pareto Archived Evolution Strategy(PAES) is an evolutionary algorithm which employs local search for the generation of new candidate solutions but utilises population information in its selection procedure. The PAES algorithm was developed with two main objectives. The first of these was that the algorithm should be strictly confined to local search i.e. it should

Algorithm 4 SPEA2 Algorithm (ZITZLER; LAUMANNS; THIELE, 2001)

1: Read N - Population size
2: Read $\bar{N}$ - Archive size
3: Read T - Maximum number of generations
4: Generate a initial population P_0
5: Create a initial archive $\bar{P}$
6: Set T to zero
7: Calculate the fitness value of individuals in P_t
8: Calculate the fitness value of individuals in $\bar{P}_t$
9: Environmental Selection - Copy all non-dominated individuals in P_t and $\bar{P}_t$ to $\bar{P}_{t+1}$
10: **if** size of $\bar{P}_{t+1}$ exceeds $\bar{N}$ **then**
11: reduce $\bar{P}_{t+1}$ by means of truncation operator
12: **else if** size of $\bar{P}_{t+1}$ less than $\bar{N}$ **then**
13: fill $\bar{P}_{t+1}$ with dominated individuals in P_t and $\bar{P}_t$
14: **end if**
15: **if** t $\gg$ T or another stopping criterion is satisfied **then**
16: set A to the set of decision vectors represented by non-dominated individuals in $\bar{P}_{t+1}$
17: Stop
18: **end if**
19: Mating selection - Perform binary tournament selection with replacement on $\bar{P}_{t+1}$ in order to fill the mating pool

use a small change (mutation) operator only, and move from a current solution to a nearby neighbour. The second objective was that the algorithm should be a true Pareto optimiser, treating all non-dominated solutions as having equal value (KNOWLES; CORNE, 1999). In PAES one parent generates by mutation one offspring. The offspring is compared with the parent. If the offspring dominates the parent, the offspring is accepted as the next parent and the iteration continues. If the parent dominates the offspring, the offspring is discarded and the new mutated solution (a new offspring) is generated. If the offspring and the parent do not dominate each other, a comparison set of previously non-dominated individuals is used (KNOWLES; CORNE, 1999)(OLTEAN; ABRAHAM; MARIO, 2005). The algorithmic framework of PAES is described in Alg. 5.

A Multiobjective Evolutionary Algorithm Based on Decomposition

The Multiobjective Evolutionary Algorithm Based on Decomposition (MOEA/D) is a multiobjective evolutionary algorithm competitive to other well-known multiobjective optimization evolutionary algorithms (MOEAs) such as NSGA-II or SPEA-2 (MICHALAK, 2014).

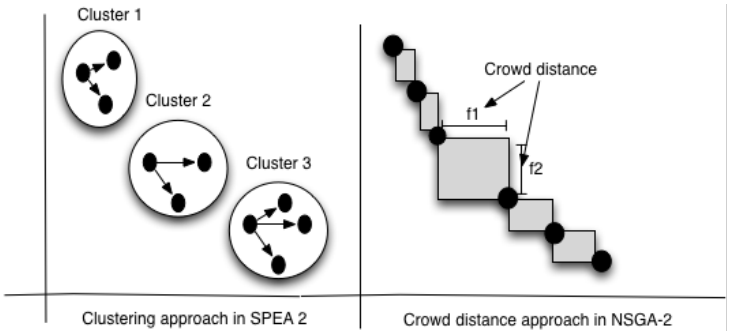


Figure 2.5: Comparison between SPEA-2 and NSGA-II (DEB; MOHAN; MISHRA, 2005)

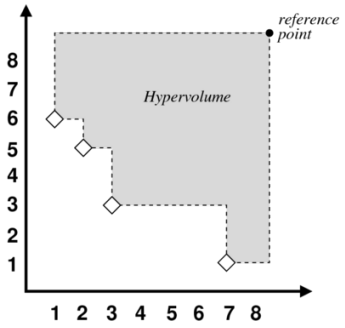


Figure 2.6: Hypervolume metric (LACOUR; KLAMROTH; FONSECA, 2015)

Algorithm 5 PAES Algorithm (KNOWLES; CORNE, 1999)(OLTEAN; ABRAHAM; MARIO, 2005)

```
Repeat until a termination criterion has been reached
Generate initial random solution c and add it to archive
Mutate c to produce m and evaluate m
if c dominates m then
    discard m
else
    if m dominates c then
        replace c with m and add m to the archive
    else
        if m is dominated by any member of the archive then
            discard m
        else
            apply test (c, m, archive) to determine which becomes the new current solution
    and whether to add m to the archive
        end if
    end if
end if
```

MOEA/D decomposes a multiobjective optimization problem into a number of scalar optimization subproblems and optimizes them simultaneously. Each subproblem is optimized by only using information from its several neighboring subproblems, which makes MOEA/D have lower computational complexity at each generation than MOGLS and non-dominated sorting genetic algorithm II (NSGA-II) (ZHANG; LI, 2007).

NSGA-II, SPEA2 and PAES algorithms do not associate each individual solution with any particular scalar optimization problem. In a scalar objective optimization problem, all the solutions can be compared based on their objective function values and the task of a scalar objective evolutionary algorithm (EA) is often to find one single optimal solution. MOEA/D explicitly decomposes the problem into scalar optimization subproblems.It solves these subproblems

simultaneously by evolving a population of solutions. At each generation, the population is composed of the best solution found so far for each subproblem. The neighborhood relations among these subproblems are defined based on the distances between their aggregation coefficient vectors (ZHANG; LI, 2007) (MCCONAGHY et al., 2011). Fig. 2.7 MOEA/D create a set of subproblems W_i. Each subproblem W_i is improved to obtain the Pareto frontier (MCCONAGHY et al., 2011).

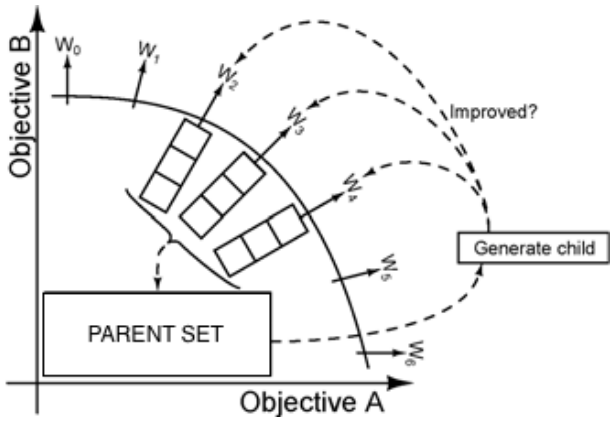


Figure 2.7: MOEA/D algorithm subproblems (MCCONAGHY et al., 2011)

The Alg. 6 presents the details of the algorithm. The algorithm includes the use of Tchebycheff approach as the decomposition method, with dynamical resource allocation, to improve the efficiency of the MOEA/D (YUEN; RAMLI, 2009).

Comparing Multi-Objective Metaheuristics

Deb states that are two orthogonal goals for any multi-objective algorithm (DEB, 2001):

- Identify solutions as close as possible to the true Pareto frontier;
- identify a diverse of sets of solutions distributed across the entire Pareto-optimal surface.

There are several metrics either closeness or diversity. Example of metrics which measure the closeness of Pareto frontier is Error ratio and Set coverage. Example of metrics which measure the diversity are the Spacing and the Spread. The Hypervolume metric measure both closeness and diversity (JANSSENS; PANGILINAN, 2010). The Hypervolume metric calculates the volume in an objective space covered by the non-dominated individuals. The hypervolume was originally proposed by Zitzler and Thiele (ZITZLER; THIELE, 1999). It is especially useful when the true Pareto-optimal solution is unknown. For each solution, a hypercube is computed from a reference point and the solution as the diagonal corners of the hypercube (Figure 2.6)

Algorithm 6 MOEA/D Algorithm

1: **Initialization**
2: Generate initial population by uniformly spreading and randomly sampling from search space
3: Calculate the reference point for the Tchebycheff approach.
4: Evaluate Objective Values
5: Selection using tournament selection method
6: Selection of mating and updating range
7: Reproduction
8: Repair
9: Update of solutions
10: **Update**
11: While (not equal to termination condition)
12: Evaluate Objective Values
13: Selection using tournament selection method
14: Selection of mating and update range
15: Reproduction
16: Repair - if the searching element is out of boundary
17: Update the solutions
18: **Stopping Criteria**
19: **if** generation is a multiplication of a pre-set value of x **then**
20: Update utility function;
21: **end if**

(JANSSENS; PANGILINAN, 2010). The reference point is found by constructing a vector of worst objects fitness value.

2.2 Reinforcement Learning

Reinforcement learning (RL) refers to both a learning problem and a sub-field 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 (SUTTON; BARTO, 2012). 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 (SUTTON; BARTO, 2012). A typical setting where Reinforcement Learning operates is shown in Figure 2.8: 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.

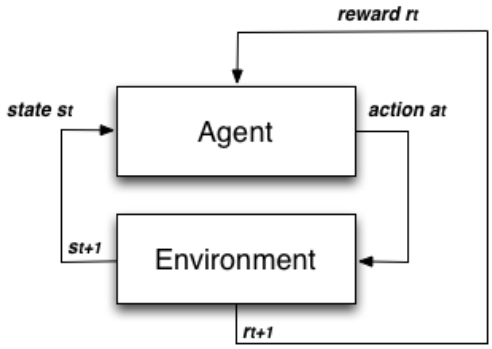


Figure 2.8: Example of interaction between some agent and the environment

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 Exploitation. 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 multi-agent learning algorithm that learns equilibrium policies in Markov games, just as Q-learning learns to optimize policies in Markov decision processes (GREENWALD; HALL; SERRANO, 2003). 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:

$$< s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, r_4, s_4 .... >, \tag{2.1}$$

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 2.9 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.

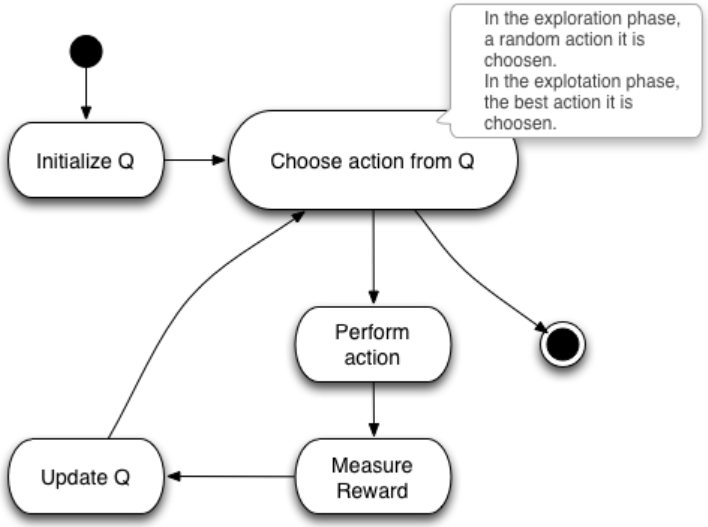


Figure 2.9: Q Learning algorithm

2.3 Summary

Metaheuristics are strategies that guide the search process to efficiently explore the search space in order to find optimal solutions. There are different ways to classify and describe metaheuristic algorithm. Algorithms working on single solutions are called trajectory methods. 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. Trajectory methods are characterized by a trajectory in the search space. Two common trajectory methods are Simulated Annealing and Tabu Search. Population-based 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 combination of one metaheuristic with components from other metaheuristics is called a hybrid metaheuristic. One of the most popular ways of metaheuristic hybridization consists in the use of trajectory methods inside population-based methods. Multi-objective optimization methods have as main purposes to minimize the distance between the non-dominated front and the Pareto optimal front and find a set of solutions that are as diverse as possible. Reinforcement learning is the problem of getting an agent to act in the unknown world so as to maximize its rewards. The next chapter presents the state-of-art of search-based stress testing.

3 SEARCH-BASED STRESS TESTING

In this chapter, we introduce the context where this thesis is inserted and discuss the relevant state-of-the-art related to Search-based Stress Testing.

3.1 Load, Performance and Stress Testing

Load, performance, and stress testing are typically done to locate bottlenecks in a system, to support a performance-tuning effort, and to collect other performance-related indicators to help stakeholders get informed about the quality of the application being tested (SANDLER; BADGETT; THOMAS, 2004) (CORPORATION, 2007).

Typically, the most common kind of performance testing for Internet applications is load testing. Application load can be assessed in a variety of ways (PERRY, 2004):

- Concurrency. Concurrency testing seeks to validate the performance of an application with a given number of concurrent interactive users (PERRY, 2004).
- Stress. Stress testing seeks to validate the performance of an application when certain aspects of the application are stretched to their maximum limits. This can include maximum number of users, and can also include maximizing table values and data values (PERRY, 2004).
- Throughput. Throughput testing seeks to validate the number of transactions to be processed by an application during a given period of time. For example, one type of throughput test might be to attempt to process 100,000 transactions in one hour (PERRY, 2004).

Performance testing aims at verifying a specified system performance. This kind of test is executed by simulating hundreds of simultaneous users or more over a defined time interval (Di Lucca; FASOLINO, 2006). The purpose of this assessment is to demonstrate that the system reaches its performance objectives (SANDLER; BADGETT; THOMAS, 2004). Term often

used interchangeably with “stress” and “load” testing. Ideally “performance” testing is defined in requirements documentation or QA or Test Plans (LEWIS; DOBBS; VEERAPILLAI, 2005).

In a load testing, the system is evaluated at predefined load levels (Di Lucca; FASOLINO, 2006). The aim of this test is to determine whether the system can reach its performance targets for availability, concurrency, throughput, and response time. Load testing is the closest to real application use (MOLYNEAUX, 2009). A typical load test can last from several hours to a few days, during which system behavior data like execution logs and various metrics are collected (AFZAL; TORKAR; FELDT, 2009).

Stress testing investigates the behavior of the system under conditions that overload its resources. The stress testing verifies the system behavior against heavy workloads (SANDLER; BADGETT; THOMAS, 2004) (LEWIS; DOBBS; VEERAPILLAI, 2005), which are executed to evaluate a system beyond its limits, validate system response in activity peaks, and verify whether the system is able to recover from these conditions. It differs from other kinds of testing in that the system is executed on or beyond its breakpoints, forcing the application or the supporting infrastructure to fail (Di Lucca; FASOLINO, 2006) (MOLYNEAUX, 2009).

The main difference between load tests, performance tests and stress tests are:

- Performance tests demonstrate that the system reaches its performance objectives.
- Load tests necessary use a load (concurrent or simultaneous users).
- Stress tests differs from other kinds of testing in that the system is executed on or beyond its break-points, forcing the application or the supporting infrastructure to fail.

3.2 Search-Based Testing

Search-based software engineering (SBSE) is the application of optimization techniques in solving software engineering problems. The applicability of optimization techniques in solving software engineering problems is suitable as these problems frequently encountered competing constraints and require near optimal solutions (AFZAL; TORKAR; FELDT, 2009) (HARMAN; JIA; ZHANG, 2015).

Search-based Software Testing (SBST) is the subarea of Search Based Software Engineering concerned with software testing. Search-based software testing is the application of meta-heuristic search techniques to generate software tests. The test adequacy criterion is transformed into a fitness function and a set of solutions in the search space are evaluated with respect to

the fitness function using a metaheuristic search technique (AFZAL; TORKAR; FELDT, 2009) (ALETI; MOSER; GRUNSKE, 2016) (HARMAN; JIA; ZHANG, 2015). A variety of metaheuristic search techniques is found to be applicable for non-functional testing including simulated annealing, tabu search and genetic algorithms.

In the academic context, a number of studies proving the efficacy of metaheuristics to automate test execution can be found in the literature. Figure 3.1 shows the growth in papers published on SBST and SBSE. The data is taken from the SBSE repository (`http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/`) and Afzal et al. (AFZAL; TORKAR; FELDT, 2009) (HARMAN; JIA; ZHANG, 2015). The aim of the SBSE repository is to contain every SBSE paper. Although no repository can guarantee 100% precision and recall, the SBSE repository has proved sufficiently usable that it has formed the basis of several other detailed analyses of the literature (HARMAN; JIA; ZHANG, 2015).

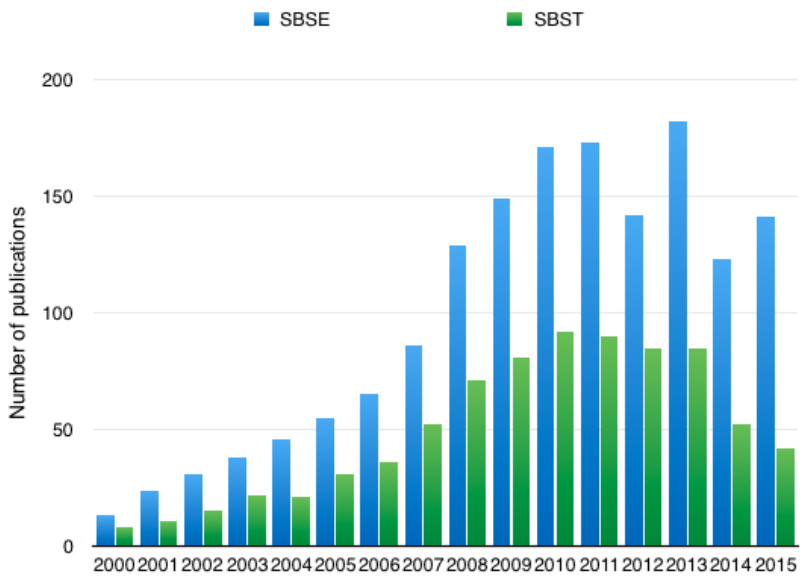


Figure 3.1: Number of publications in SBSE and SBST by year (AFZAL; TORKAR; FELDT, 2009) (HARMAN; JIA; ZHANG, 2015)

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. Fig. 3.2 shows the non-functional SBST by year, data comes from the Harman et al., Afzal et al. and the SBSE repository (ALETI; MOSER; GRUNSKE, 2016) (HARMAN; JIA; ZHANG, 2015).

There are many kinds of non-functional search based tests (AFZAL; TORKAR; FELDT, 2009):

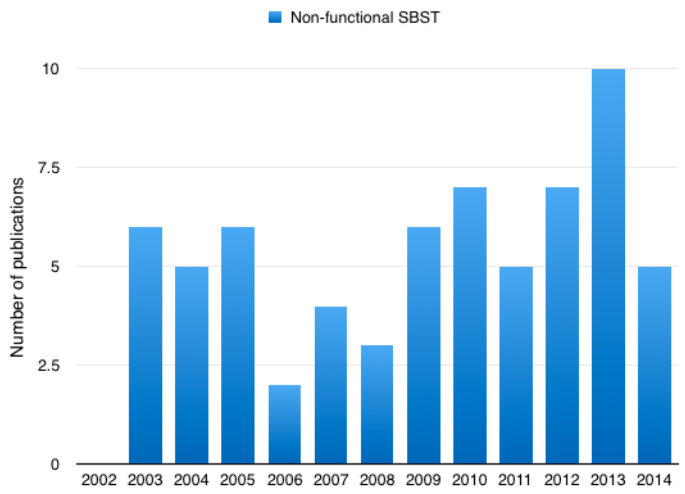


Figure 3.2: Number of publications in non-functional SBST by year (AFZAL; TORKAR; FELDT, 2009) (HARMAN; JIA; ZHANG, 2015)

- 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 the 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 is 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. However, most research studies are limited to making prototypes (AFZAL; TORKAR; FELDT, 2009). Fig. 3.3 shows a comparison between the range of metaheuristics and the type of non-functional search based test.The Data comes from Afzal et al. (AFZAL; TORKAR; FELDT, 2009). Afzal's work was added to some of the latest research in this area ((GAROUSI, 2006) (GAROUSI, 2010) (Di Alesio et al., 2013) (ALESIO et al., 2014) (ALESIO et al., 2015) (GOIS et al., 2016)). The thesis focus is use hybrid and multiobjective metaheuristics in Quality of Service and Execution Time tests (Search-based stress testing). There is a great difficulty in comparing

some approaches present in the state of the art, due to the lack of availability of the tools used. The present research intends to compare the proposed approach with other methods based on the use of single or multiobjective metaheuristics.

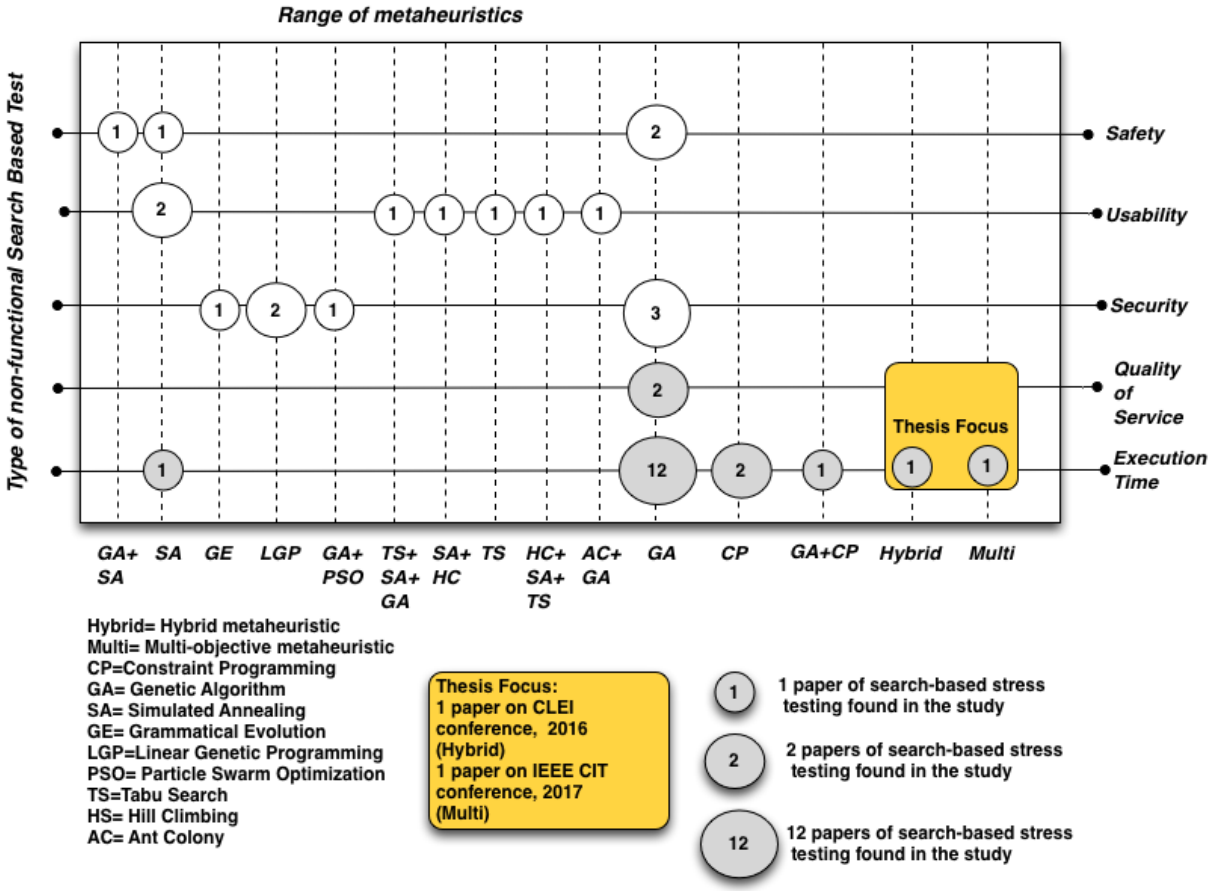


Figure 3.3: Range of metaheuristics by type of non-functional Search Based Test(AFZAL; TORKAR; FELDT, 2009).

3.3 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 (SULLIVAN et al., 1998). 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 (AFZAL; TORKAR; FELDT, 2009).

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 (AFZAL; TORKAR; FELDT, 2009)

(TRACEY, 2000). 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 from each platform. Execution time measurement is a non-deterministic approach, in which there is no guarantee of obtaining the same test inputs (AFZAL; TORKAR; FELDT, 2009). However, stress testing where testers have no access to the production environment should be measured by the execution time measurement (MOLYNEAUX, 2009) (AFZAL; TORKAR; FELDT, 2009).

Table 3.1 shows a comparison between the research studies on load, performance, and stress tests presented by Afzal et al. (AFZAL; TORKAR; FELDT, 2009). Afzal's work was added with some of the latest research in this area ((GAROUSI, 2006) (GAROUSI, 2010) (Di Alesio et al., 2013) (ALESIO et al., 2014) (ALESIO et al., 2015) (GOIS et al., 2016)). The columns represent the type of tool used (prototype or functional tool), and the rows represent the metaheuristic approach used in 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 or Search-Based Stress Testing on non Safety-critical systems.

3.3.1 Search-Based Stress Testing 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 (Di Alesio et al., 2013).

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, and other synchronization of parallel processes are of major importance for the correct function of real-time systems (J. Wegener, K. Grimm, M. Grochtmann, H. Sthamer, 1996) .

Table 3.1: 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 + Multi-objective heuristics NSGA-II, SPEA2 PAES,MOEA/D			Our approach (GOIS et al., 2016) (GOIS; PORFIRIO; COELHO, 2017)
GA	Alander et al.,1998 (ALANDER; MANTERE; TURUNEN, 1998) Wegener et al., 1996 and 1997 (WEGENER et al., 1997) (J. Wegener, K. Grimm, M. Grochtmann, H. Sthamer, 1996) (SULLIVAN et al., 1998) (BRIAND; LABICHE; SHOUSHA, 2005) Canfora et al., 2005 (CANFORA et al., 2005)	Wegener and Grochtmann, 1998 (WEGENER; GROCHTMANN, 1998) Mueller et al., 1998 (MUELLER; WEGENER, 1998) Puschner et al. (PUSCHNER; NOSSAL, 1998) Wegener et al., 2000 (Wegener, Joachim and Pitschinetz, Roman and Sthamer, 2000) Gro et al., 2000 (GROSS; JONES; EYRES, 2000)	Di Penta et al., 2007 (PENTA; CANFORA; ESPOSITO, 2007) Garoussi, 2006 (GAROUSI, 2006) Garousi, 2008 (GAROUSI, 2008) Garousi, 2010 (GAROUSI, 2010)
Simulated Annealing (SA)			Tracey, 1998 (TRACEY; CLARK; MANDER, 1998)
Constraint Programming			Di Alesio et al., 2014 (ALESIO et al., 2014) Di Alesio et al., 2013 (Di Alesio et al., 2013)
GA + Constraint Programming			Di Alesio et al., 2015 (ALESIO et al., 2015)
Customized Algorithm		Pohlheim, 1999 (POHLHEIM; CONRAD; GRIEP, 2005)	
SIBEA Multi-object algorithm	Woehrle, 2012 (WOEHRLE, 2012)		

The concurrent nature of embedded software makes the order of external events triggering the system tasks are 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 (Di Alesio

et al., 2013). 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 (SHOUSHA, 2003).

The main goal of Search-Based Stress testing of Safety-critical systems is finding a combination of inputs that cause the system to delay task completion to the greatest extent possible. The followed approaches use metaheuristics to discover the worst-case execution times. Wegener et al. (WEGENER et al., 1997) used GAs to search for input situations that produce very long or very short execution times. The fitness function used was the execution time of an individual measured in micro seconds (WEGENER et al., 1997). Alander et al. (ALANDER; MANTERE; TURUNEN, 1998) performed experiments in a simulator environment to measure extreme response times 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 (ALANDER; MANTERE; TURUNEN, 1998).

Wegener and Grochtmann performed an experiment to compare GA with random testing. The fitness function used was the execution duration 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 (J. Wegener, K. Grimm, M. Grochtmann, H. Sthamer, 1996) (WEGENER; GROCHTMANN, 1998). Gro et. al. (GROSS; JONES; EYRES, 2000) 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 (GROSS; JONES; EYRES, 2000). Briand et al. (BRIAND; LABICHE; SHOUSHA, 2005) 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 a GA. Two case studies were conducted and results illustrated that RTTT was a useful tool to stress a system under test (BRIAND; LABICHE; SHOUSHA, 2005).

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 (POHLHEIM; CONRAD; GRIEP, 2005). Garousi presented a stress test methodology aimed at increasing the 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 (GAROUSI, 2006). Alesio, Nejati and Briand 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 GAs. In particular, the study compares CP and GA in five case studies for efficiency, effectiveness, and scalability. The experimental results show that, on the larger 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 (Di Alesio et al., 2013).

Alesio describes stress test case generation as a search problem over the space of task arrival times. The research locates the worst-case scenarios maximizing deadline misses where each scenario characterizes a test case. The paper combines two strategies, GA and 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 (ALESIO et al., 2015).

3.3.2 Search-Based Stress Testing on non Safety-critical Systems

Usually, the application of Search-Based Stress Testing on non safety-critical systems deals with the generation of test cases that cause Service Level Agreement (SLA) violations.

Tracey et al. (TRACEY; CLARK; MANDER, 1998) 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 for a detailed comparison of various optimization techniques to explore the worst-case execution time (WCET) and the best-case execution times (BCET) of the system under test (TRACEY; CLARK; MANDER, 1998).

Di Penta et al. (PENTA; CANFORA; ESPOSITO, 2007) used GA to create test data that violated quality of service (QoS) constraints, causing SLA violations. The generated test data included combinations of inputs. The approach was applied to two case studies. The first case study was an audio processing workflow, and the second case study, a service producing charts (PENTA; CANFORA; ESPOSITO, 2007).

Gois et al. propose 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 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 the previously established six generations (GOIS et al., 2016).

Gois et al. verify the use of a multi-objective algorithm in search-based stress testing. The NSGA-II algorithm was implemented in the IAdapter tool using the jMetal framework. The experiment uses the NSGA-II algorithm to discover application scenarios where there is a high response time for a small number of users. The experiment found 9 optimal workloads that present a lower number of users with high response times (GOIS; PORFIRIO; COELHO, 2017).

3.4 Similar approaches

This research and Alesio's approach (ALESIO et al., 2015) use a hybrid approach with a functional tool in different contexts. Table 3.2 presents the main differences between Alesio's and this thesis's approaches. Whereas the present research uses an approach based on usage scenarios performing tests on an application installed in an available environment, Alesio uses sequence diagrams to select for the arrival time of tasks in systems from safety-critical domains.

Woehrle and this thesis use multi-objective algorithms to stress testing in different contexts. Woehrle formulate stress testing of protocol stacks on specific topologies as a multi-objective optimization problem and use an evolutionary algorithm for finding a set of small topologies that particularly stress the protocol stack of a wireless network (WOEHRLE, 2012).

3.5 Summary

The stress testing verifies the system behavior against heavy workloads, which are executed to evaluate a system beyond its limits, validate system response in activity peaks, and verify whether the system is able to recover from these conditions. Search-based software engineering

Table 3.2: Main differences between Alesio's (ALESIO et al., 2015) and IAdapter's approaches

	Alesio et al. (ALESIO et al., 2015)	This thesis
Metaheuristcs	GA+ Constraint Programming	GA+SA+ Tabu Search
Inputs	Design Model (Time and Concurrency Information)	Number of Users and Test scenarios
Main Objective	Find task arrival times of aperiodic tasks that maximizing deadline misses	Find the number of users and test scenarios that maximizing the response time
Main Application	Systems from safety-critical domains	Web and Mobile applications

(SBSE) is the application of optimization techniques in solving software engineering problems. The applicability of optimization techniques in solving software engineering problems is suitable as these problems frequently encounter competing constraints and require near optimal solutions. Search-based software testing is the application of metaheuristic search techniques to generate software tests. The test adequacy criterion is transformed into a fitness function and a set of solutions in the search space is evaluated with respect to the fitness function using a metaheuristic search technique.

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 variety of metaheuristic search techniques is 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. However, most research studies are limited to making prototypes. 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. 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. 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. 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.

The studies can be divided into two main groups: Search-Based Stress Testing on Safety-critical systems or Search-Based Stress Testing on non Safety-critical systems. Domains such as avionics feature safety-critical systems, whose failure could lead to 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.

Usually, the application of Search-Based Stress Testing on non safety-critical systems deals with the generation of test cases that cause Service Level Agreement (SLA) violations. Tracey et al. used simulated annealing (SA) to test four simple programs. The results of the research presented that the use of SA was more effective with a larger parameter space. Di Penta et al. used GA to create test data that violated quality of service (QoS) constraints, causing SLA violations. The approach was applied to two case studies. The first case study was an audio processing workflow, and the second case study, a service producing charts. The next chapter presents the first contribution of this thesis.

4 STRESS SEARCH BASED TESTING USING HYBRID METAHEURISTIC

This chapter presents a study of the use a hybrid algorithm in search-based stress testing. The research uses genetic algorithms, tabu search, and simulated annealing in two different approaches. The study initially investigated the use of these three algorithms independently. Subsequently, the study will focus on 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. 4.1). The second approach uses the algorithms in a collaborative mode (hybrid metaheuristic). In this approach, the three algorithms share their best individuals found (Fig. 4.2).

The chapter also presents three experiments. The first experiment aimed to perform performance, load, and stress testing on a simulated component.The second experiment was performed with a Moodle application. The third experiment carried out to verify the best case scenarios found by the hybrid metaheuristic approach. 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 4.3 -❶). The first position represents a metadata with the name of an individual. The next positions represent 10 scenarios and their numbers of users (Figure 4.3 -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. 4.3 presents the solution representation and an example using the crossover opera-

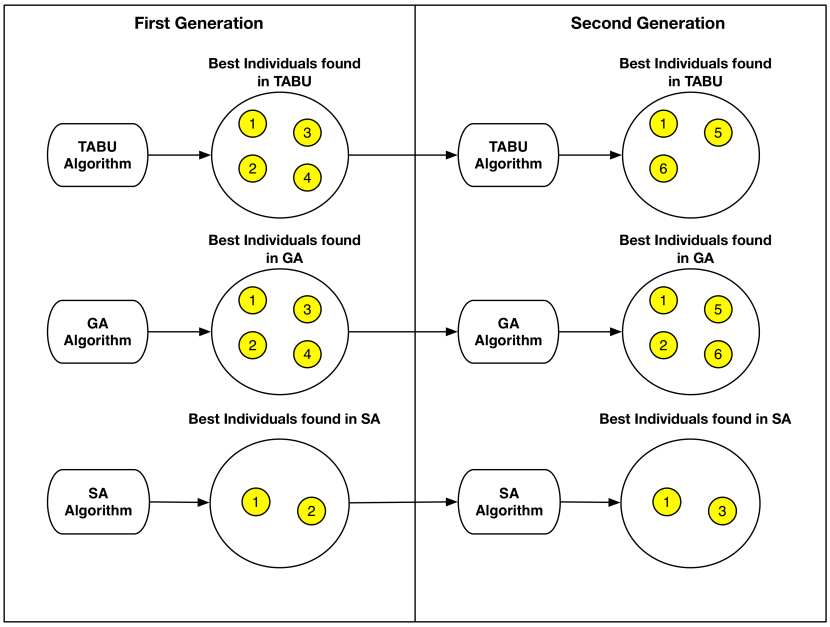


Figure 4.1: Use of the algorithms independently (GOIS et al., 2016)

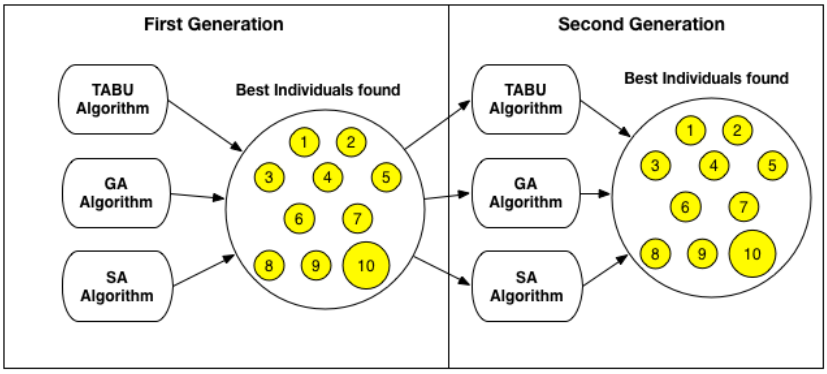


Figure 4.2: Use of the algorithms collaboratively (GOIS et al., 2016)

tion. In the example, solution 1 (Figure 4.3 -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 4.3 -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 4.3 -❺). Figure. 4.3 -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.

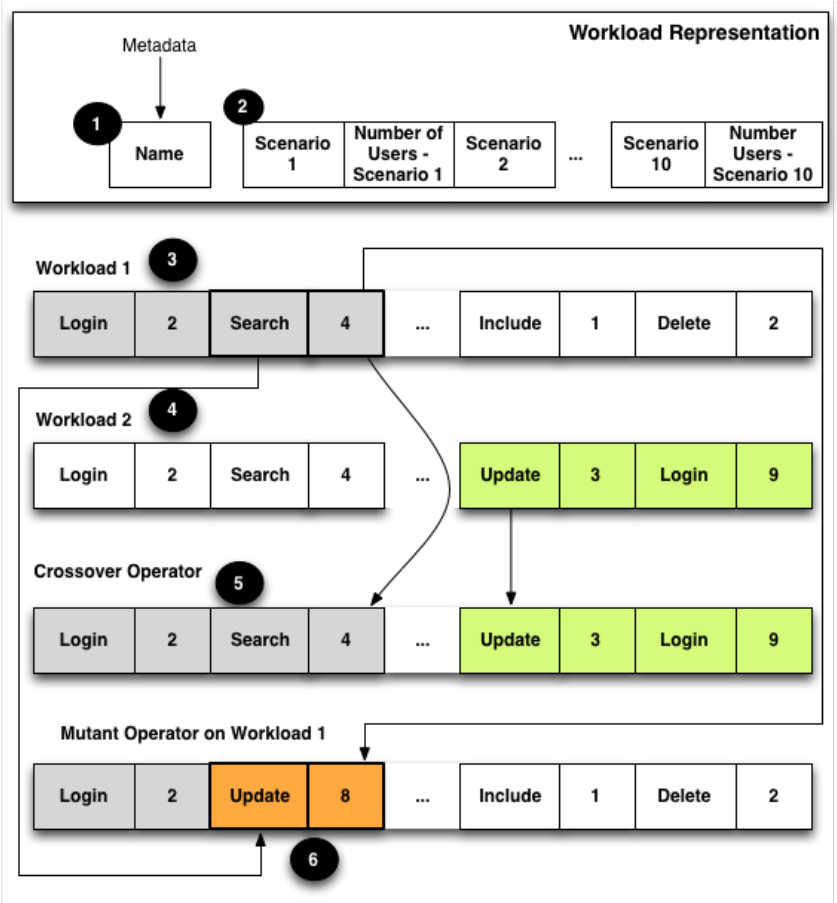


Figure 4.3: Solution representation, crossover and neighborhood operators (GOIS et al., 2016)

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:

$$\begin{aligned} fit = {} & numberOfUsersWeight * numberOfUsers \\ & - 90percentileweight * 90percentiletime \\ & - 80percentileweight * 80percentiletime \\ & - 70percentileweight * 70percentiletime \\ & - maxResponseWeight * maxResponseTime \\ & - penalty \end{aligned} \tag{4.1}$$

The users and response time factors were chosen because they are common units of measurement in load test tools (SANDLER; BADGETT; THOMAS, 2004). The proposed solution's fitness function uses a series of manually adjustable user-defined weights (90percentileweight, 80percentileweight, 70percentileweight, maxResponseWeight, and numberOfUsersWeight). These weights make it possible to customize the search plugin's functionality. A penalty is applied when the response time of an application under test runs longer than the service level. The penalty is calculated by the following equation:

$$\begin{aligned} penalty &= 100 * \Delta \\ \Delta &= (t_{CurrentResponseTime} - t_{MaximumResponseTimeExpected}) \end{aligned} \tag{4.2}$$

4.4 Experiments with Hybrid Algorithm

This section presents three experiments. The first one was performed on an emulated component, the second one was performed using an installed Moodle application and the third one was performed using four antipatterns. The experiments used the following fitness function:

$$
\begin{aligned}
fit = 0.9 * 90percentiletime \\
+0.1 * 80percentiletime \\
+0.1 * 70percentiletime + \\
0.1 * maxResponseTime + \\
0.2 * numberOfUsers - penalty
\end{aligned}
\tag{4.3}
$$

This fitness function has manually adjustable user-defined weights filled out. This fitness function intended to find individuals with the highest percentile of 90%, followed by individuals with a higher percentile time of 80% and 70%, maximum response time, and a number of users.

The first experiment ran for 27 generations, and the second experiment performed 6 generations, with 300 executions by generation (100 times for each algorithm), generating 300 new individuals. The experiments used an initial population of 100 individuals. The genetic algorithm used the top 10 individuals from each generation in the crossover operation. The Tabu list was configured with the size of 10 individuals and expired every 2 generations. The mutation operation was applied to 10% of the population on each generation.

4.4.1 First Experiment: Emulated Class Test

The first experiment aimed to perform performance, load, and stress testing on a simulated component. The purpose of using a simulated component was to be able to perform a greater number of generations in a shorter time available and eliminate variables such as the use of databases and application servers. The first experiment used a test class named SimulateConcurrentAccess. This class has a static variable named x and a set of methods that use the variable in a synchronized context (Listing 4.1). The experiment was executed using the JMeter Java Request Sampler Component with IAdapter.

Fig.4.4 presents the best results in 27 generations applied in the first experiment. The figure shows the results obtained with the algorithms with and without collaboration. The x axis represents the generation number, and the y axis represents the best fitness value obtained until the current generation. A higher value in the figure means that the scenario has a greater response time by the application under test. The results of the experiment showed that the use of cooperation between the three algorithms resulted in finding the individuals with better fitness values.

Table 4.1 presents the results obtained by the hybrid metaheuristic (HM) approach, genetic algorithm (GA), simulated annealing (SA), and Tabu search (TS) from 27 generations in the

Listing 4.1 SimulateConcurrentAccess class

```
1:  public class SimulateConcurrentAccess {
2:    @Test
3:    public void firstScenario() {
4:      synchronized (StaticClass.class) {
5:        for (int i = 0; i <= 1000; i++) {
6:          StaticClass.x += i;
7:        }
8:        StaticClass.x = 0;
9:      }
10:   }
11:
12:     @Test
13:   public void secondScenario() {
14:     synchronized (StaticClass.class) {
15:       for (int i = 0; i <= 2000; i++) {
16:         StaticClass.x += i;
17:       }
18:       StaticClass.x = 0;
19:     }
20:   }
```

first experiment. The values are the maximum fitness value obtained by each algorithm.

The signed-rank Wilcoxon non-parametrical procedure was used for comparing the results with Z-value and W-value. The significant level adopted was 0.05. The Z-value obtained was -2.2736 and the p-value was 0.0232. The W-value obtained was 78. The critical value of W for N = 25 at p<= 0.05 was 89.The result was significant at p<= 0.05. The procedure showed that there was a significant improvement in the results with the collaborative approach.

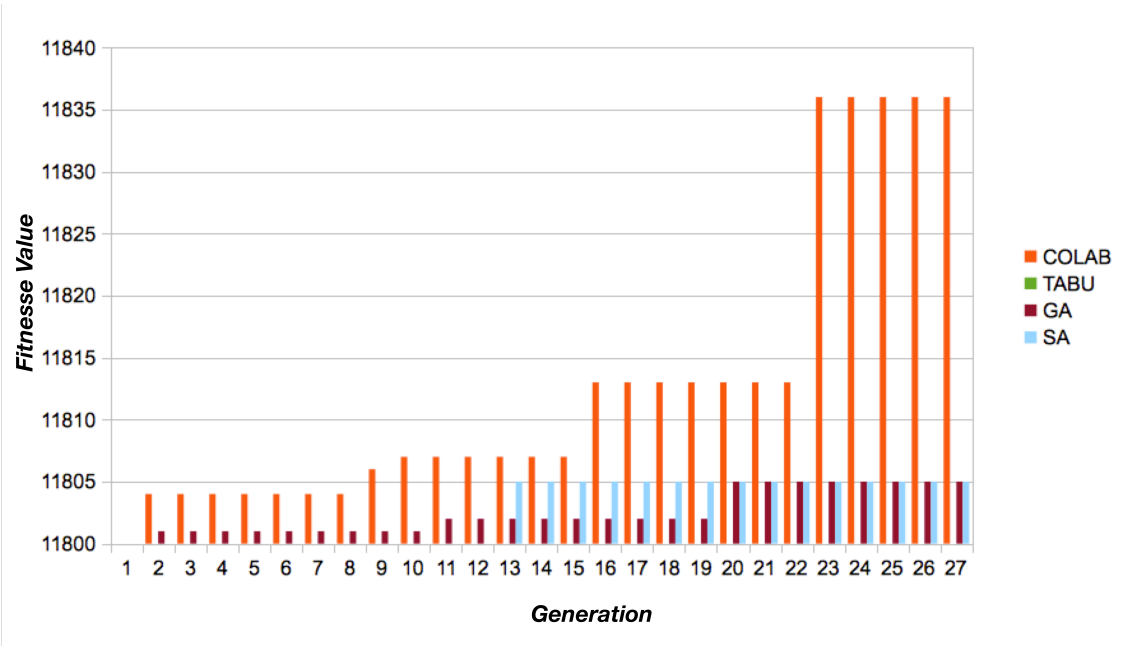
4.4.2 Second Experiment: Moodle Application Test

The second experiment used a Moodle application installed on a machine with 500 GB of hard disk space and 8 GB of memory. The study used six application scenarios:

- PostDeleteMessage: This scenario posts and deletes messages in the Moodle application.
- MyHome: This scenario accesses the homepage of the user's application.
- Login: This scenario is responsible for user authentication by the application.
- Notifications: This scenario involves entering the notification page of each user.
- Start Page: This scenario shows the initial start page of the application.
- Badge: This scenario involves entering the badge page.

The maximum tolerated response time in the test was 30 seconds. Any individuals who obtained a time longer than the stipulated maximum time suffered penalties. The whole process

Figure 4.4: Best results obtained in 27 generations



of stress and performance tests, which took 3 days and about 1800 executions, was carried out without the need for monitoring by a test designer. The tool automatically selected the next scenarios to be run up to the limit of six generations previously established.

Table 4.2 presents the maximum fitness value obtained by the hybrid metaheuristic (HM) approach, genetic algorithm (GA), simulated annealing (SA), and Tabu search (TS) in each generation.

The small number of samples of the experiment is insufficient to give a statistical significance to the results of the Wilcoxon procedure. However, it is noted that, in four of six generations, the collaborative approach presented the best values. The experiment succeeded in finding 29 individuals whose maximum time expected by the application was obtained. Table 4.3 shows an example of the six individuals with the highest fitness values in the second experiment. The table shows the fitness value (Fit); the name of the scenario (Scenario); the number of users (Users); and the percentiles of 90%, 80%, and 70% (90per, 80per and 70per) in seconds.

Table 4.4 presents the percentage of genes in all test scenarios by generation with and without collaboration. Most of the genes converged to the MyHome feature, which had the highest application response time.

Table 4.1: Maximum value of the fitness function by algorithm

GEN	HM	TS	GA	SA
1	11238	11238	11238	11238
2	11804	11596	11801	10677
3	11787	8932	8411	10869
4	11723	9753	9611	10760
5	8164	9780	10738	4794
6	11802	9781	11086	6120
7	9985	5782	11272	11798
8	11803	11749	10084	11309
9	11806	7284	11633	10766
10	11807	9386	11717	4557
11	11802	9653	11802	11151
12	11807	10594	11793	9434
13	11802	10848	10382	11805
14	11801	11551	7219	10237
15	11807	1701	7189	9338
16	11813	6203	11758	5321
17	11805	10720	10805	11748
18	9600	6371	11698	7818
19	11733	8160	11648	11509
20	9589	9428	11805	4813
21	11800	9463	11798	10801
22	11805	11799	11804	6029
23	11836	11655	11800	3579
24	11805	11512	11803	5761
25	11804	11573	11802	9680
26	11800	11575	11403	9388
27	11805	10691	11745	9465

Table 4.2: Results obtained from the second experiment

GEN	HM	TS	GA	SA
1	32242	32242	32242	32242
2	34599	32443	26290	35635
3	35800	34896	34584	34248
4	35782	34912	32689	25753
5	35611	31833	34631	8366
6	35362	35041	33397	9706

4.4.3 Third Experiment: Anti-patterns

In this subsection, We present the results of the experiment which we carried out to verify the best case scenarios found by the hybrid metaheuristic approach. We conducted the experiment in two steps in order to verify the effectiveness of the hybrid algorithm. The first step uses

Table 4.3: Example of individuals obtained in the second experiment

Id	Fit	Scenario	Users	90per	80per	70per
1	35800	MyHome	31	30	29	10
		Badges	4			
2	35795	MyHome	30	30	29	10
		Notifications	2			
		Badges	2			
3	35782	MyHome	32	30	29	10
		Badges	3			
4	35773	MyHome	22	30	29	10
		Notifications	6			
		Badges	9			
5	35771	MyHome	28	30	29	9
		Badges	6			
6	35683	MyHome	27	30	29	8
		Badges	10			

Table 4.4: Percentage of genes in each scenario by generation

Gen/ Scenarios	**Non collaboration approach**						
	Initial	1	2	3	4	5	6
Badges	20	18	16	24	15	16	17
MyHome	**15**	**59**	**55**	**48**	**53**	**50**	**52**
StartPage	15	10	12	11	20	18	19
Notifications	25	5	11	10	9	10	9
Post	8	3	1	3	1	2	1
Login	17	5	5	4	2	4	2
	Collaboration approach						
Badges	20	29	16	25	9	16	9
MyHome	**15**	**29**	**69**	**49**	**74**	**66**	**76**
StartPage	15	22	10	21	10	10	8
Nofications	25	10	1	1	2	1	3
Post	8	2	1	1	1	2	1
Login	17	8	3	3	4	5	3

The Ramp and Circuitous Treasure Hunt anti-pattern. The second step uses The Tower Babel and Unbalanced Processing antipattern. Each step uses two different anti-pattern and happy scenarios. The experiment ran for 17 generations. The experiment 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 experiment uses tabu search, genetic algorithms and the hybrid metaheuristic approach proposed by Gois et al. (GOIS et al., 2016). The objective function applied is intended to maximize the number of users and minimize the response time of the

scenarios being tested (Best Case Scenarios). In this experiment, better fitness values means to find scenarios with more users and lower values of a response time. A penalty is applied when the response time is greater than the maximum response time expected.

4.4.4 Experiment Research Questions

The following research question is addressed:

- Does the Hybrid algorithm finds scenarios without the antipatterns presented?

4.4.5 Variables

The independent variables are the test scenarios (antipatterns and happy scenarios). The dependent variables are: the number of antipatterns found in best workloads and the metaheuristic with the best fitness value.

4.4.6 Experiment Hypotheses

With regard to the antipatterns found by hybrid metaheuristic:

- H_0 (null hypothesis):the best workloads found in the experiments contain antipatterns
- H_1 : the best workloads found in the experiments do not contain antipatterns.

4.4.7 The Ramp and Circuitous Treasure Hunt step

The experiment was carried out for 8 continuous hours. All tests in the experiment were conducted without the need of a tester, automating the process of executing and designing performance test scenarios.In this experiment, Scenarios were generated with the Ramp and Circuitous Treasure antipattern as well as scenarios with Happy Scenario 1, Happy Scenario 2 and mixed scenarios. Figure 4.5 present the fitness value obtained by each metaheuristic.

Table 4.5 shows 4 best individuals found in the experiment by hybrid algorithm. None of the best individuals has one of the antipatterns used in the experiment, excluding the scenarios with antipatterns.

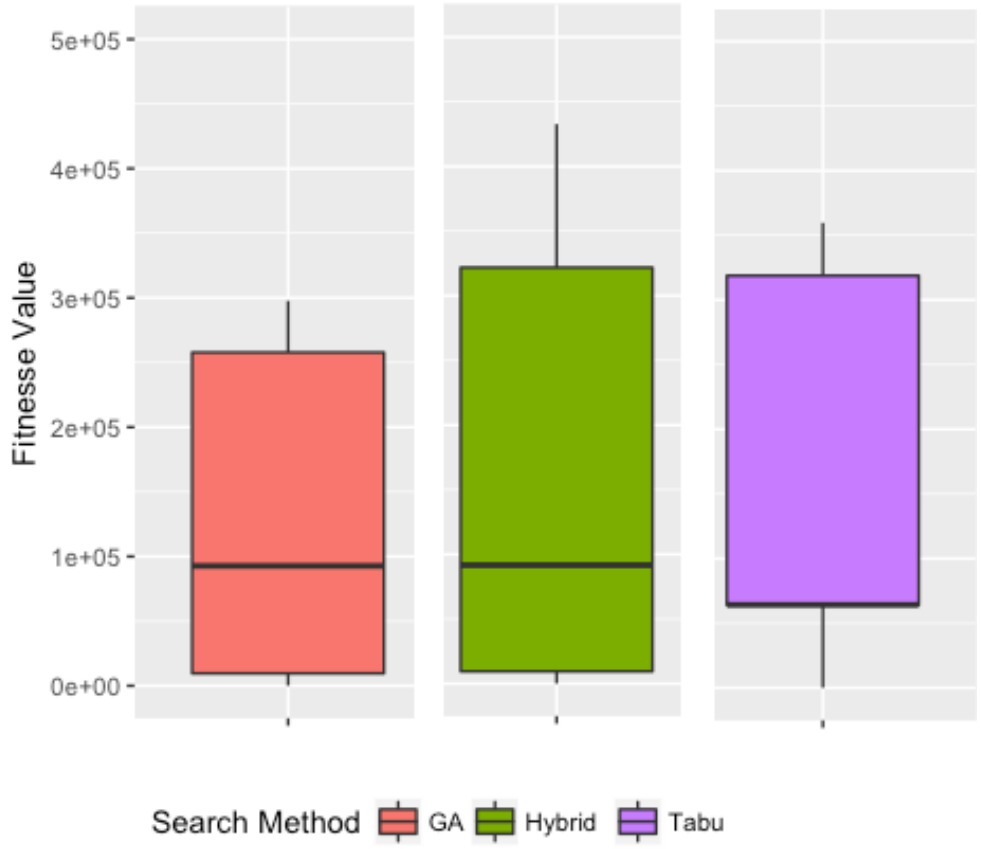


Figure 4.5: Average, median, maximum and minimal fitness value by Search Method

Table 4.5: Best individuals found in the first experiment

Metaheur.	Gen.	Users	Fit	Scenarios
Hybrid	17	145	432760	Happy 1 & 2
Hybrid	17	145	432740	Happy 1 & 2
Hybrid	17	146	431760	Happy 1 & 2
Hybrid	16	143	426740	Happy 1 & 2

4.4.8 The Tower Babel and Unbalanced Processing step

The experiment was carried out for 6 continuous hours. In this experiment, Scenarios were generated with Tower Babel and Unbalanced Processing anti-pattern as well as scenarios with Happy Scenario 1, Happy Scenario 2 and mixed scenarios.

Table 4.6 shows the 4 best workloads found in the second experiment. Despite the fact of doing 300 conversions of the JSON format to XML. The antipattern implementation does not return a higher response time than happy paths. While happy paths return from 10 to 15 seconds from a single user, Tower Babel antipattern has a response time of 10 to 29 seconds. None of the best individuals found implements the Unbalanced Processing antipattern.

In the second experiment, the metaheuristics converged to scenarios with a happy path and

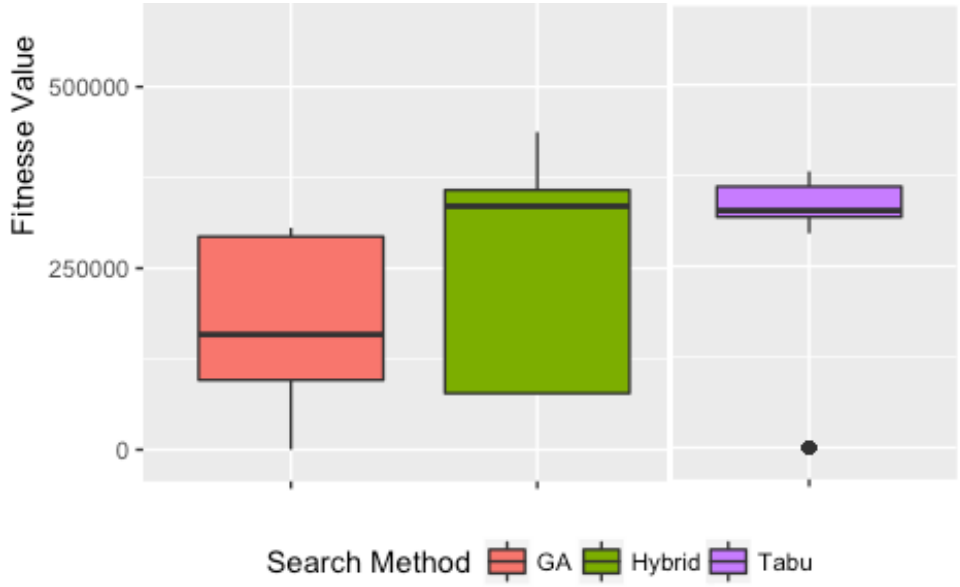


Figure 4.6: Finesse value by generation in all tests

Table 4.6: Best individuals found by hybrid algorithm in the second experiment

Metaheur.	Gen.	Users	Fit	Scenarios
Hybrid	17	148	437780	Happy 1,2 & Tower
Hybrid	17	145	432740	Happy 1,2 & Tower
Hybrid	16	146	431800	Happy 1,2 & Tower
Hybrid	17	145	428780	Happy 1,2 & Tower

Tower Babel anti-pattern, excluding the scenarios with Unbalanced Processing anti-pattern. The hybrid metaheuristic returned individuals with higher fitness scores. The SA algorithm obtained the worst fitness values.

4.4.9 Threats to validity

- Construct Validity: In this experiment, we just evaluate the use of four antipatterns. However, several anti-patterns could be applied. The common representation and the strategies used for crossover and neighborhood operators need a better design, using an abstraction pattern to contemplate a major number of possible solutions.

- Conclusion Validity: The Tower Babel anti-pattern was not excluded by the metaheuristics used in the experiment, requiring new studies with new approaches and experiments.

4.5 Conclusion

This chapter presented a hybrid metaheuristic approach for use in stress testing. Three experiments were performed to validate the solution. The first experiment was performed on an emulated component. The second experiment was performed using an installed Moodle application. The collaborative approach obtained better fit values in both first two experiments.

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

In the first experiment, the signed-rank Wilcoxon non-parametrical procedure was used for comparing the results. The significant level adopted was 0.05. The procedure showed that there was a significant improvement in the results with the Hybrid Metaheuristic approach.

In the second experiment, the whole process of stress and performance tests, which took 3 days and about 1800 executions, was carried out without the need for monitoring by a test designer. The tool automatically selected the next scenarios to be run up to the limit of six generations previously established.

The third experiment was conducted to validate the use of anti-patterns with the Hybrid algorithm. The experiments use genetic, algorithms, tabu search, simulated annealing and the hybrid algorithm. We conducted the experiment in two steps in order to verify the effectiveness of the hybrid algorithm. In the first step, none of the best workloads has one of the anti-pattern scenarios. In the second step, the metaheuristics converged to scenarios with a happy path and Tower Babel antipattern, excluding the scenarios with Unbalanced Processing anti-pattern. In both experiments, the hybrid metaheuristic returned individuals with higher fitness scores.

5 STRESS SEARCH-BASED TESTING USING HYBRIDQ APPROACH

This chapter presents a reinforcement learning approach 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 that HybridQ is more expensive than Hybrid because q-learning. The research has 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 exploitation. The following subsections show details of the exploration and exploitation phases and the integration between metaheuristics and the Q-Learning algorithm.

5.0.1 Exploration phase

The exploration phase uses a markov model, as shown in Fig. 5.1, 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 allowed, between 0.8 and 1.2 times the maximum response time allowed or less than 0.8 times the maximum response time allowed. The values of 1.2 and 0.8 were chosen 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 5.1), one for each state. The selection of which table to use depends on the response time of the application.

Algorithm 8 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

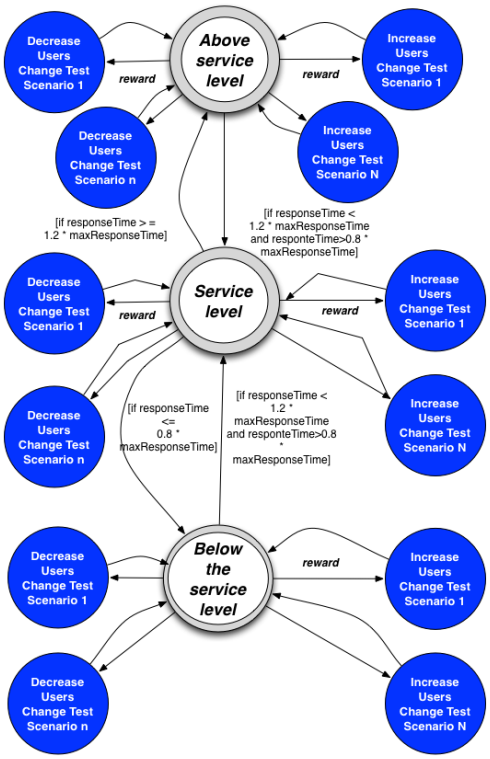


Figure 5.1: Markov Decision Process used by HybridQ

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 exploitation phase. Table 5.1 presents hypothetical Q-values for a test. In Table 5.1, 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.

5.0.2 Exploitation phase

The main objective of the exploitation phase is to choose the best neighboring solution based on the Q value. The research expected that Q-Learning improve Hybrid algorithm re-

Algorithm 7 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* maxResponse-
   Time then
5:     return qTableServiceLevel
6: end if
7: if responseTime > 1.2 * maxResponseTime then
8:     return qTableAboveServiceLevel
9: end if
```

Algorithm 8 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)
```

placing the random characteristic of the tabu search, simulated annealing and genetic algorithms operators by the direction given by the q-learning in the exploration phase. Algorithm 9 presents the main steps of exploitation phase. In first line, the algorithm gets the original genome. In lines 2 to 11, HybridQ gets the maximum, the second maximum or the third maximum q value, depending on the random value of the random variable.The algorithm chooses one of the three largest values of q. The variation of the highest values was inserted in the algorithm to escape the local optimals. In line 12, the algorithm gets the key value in Table that have the maximum q value. In line number 13, 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. Finally, the test scenario is changed and the new genome is returned.

Table 5.1: Hypothetical MDP Q-values

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

5.0.3 Integration between metaheuristics and the Q-Learning algorithm

The Q-learning algorithm is used by Tabu Search or Simulated Annealing to obtain the neighbors and in the mutation operator of the genetic algorithm. Unlike the traditional processes of obtaining neighboring solutions such as random change and permutation, the decision to change a genome gene is made from the action that has the highest value of Q. Fig. 5.2 presents how one of the neighbors of a test is generated using Q-Learning in IAdapter. The solution uses a service called Q-Neighborhood Service to generate the neighbor from the action that has the highest value of Q.

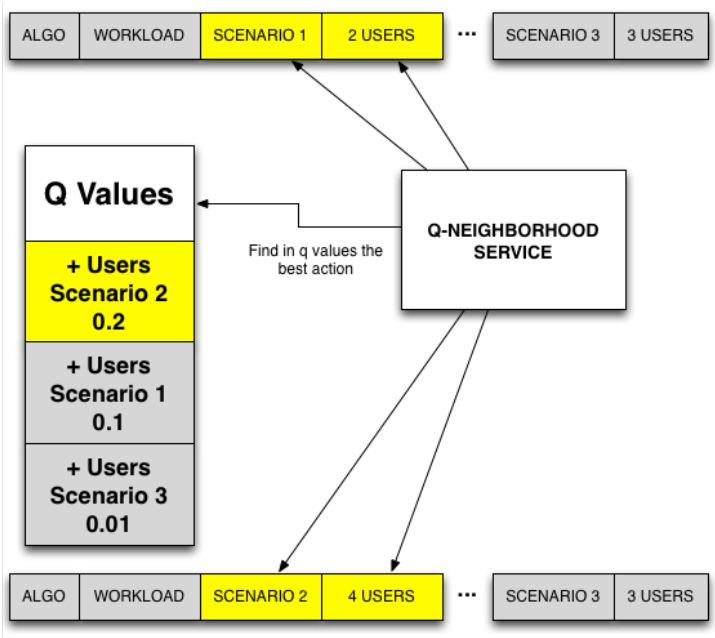


Figure 5.2: HybridQ Neighborhood Service

Algorithm 9 HybridQ exploitation phase

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

5.1 Experiment with HybridQ Algorithm

We conducted one experiment in order to verify the effectiveness of the HybridQ. The iterated racing procedure (irace) was applied as an automatic algorithm configuration 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 (Manuel López-Ibãnez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, 2016). The best parameters obtained 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. The experiment ran for 16 generations in an docker environment on a server with 16 Gb of memory and 500 Gb hard disk. The experiment 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 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 experiment used the following fitness (goal) function:

$$\begin{aligned} fitness = {} & 20 * 90 percentiletime \\ & 20 * 80 percentiletime \\ & 20 * 70 percentiletime \\ & 20 * maxResponseTime \\ & -penalty \end{aligned} \tag{5.1}$$

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

5.1.1 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?

5.1.2 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.

5.1.3 Experiment Hypotheses

- With regard to the optimal solution found by each algorithm:
  - $H1_0$ (null hypothesis) : The HybridQ does not find better solution than the other metaheuristic approaches.
  - $H1_1$: The HybridQ finds better solution than the one discovered by other metaheuristic approaches.

- With regard to the time consumed to find the optimal solution of each algorithm:
  - $H2_0$ (null hypothesis) : The HybridQ algorithm realizes more requests than the other algorithms in the experiments performed.
  - $H2_1$: The HybridQ algorithm does not realize more requests than the other algorithms in the executed experiments.

- With regard to the number of requests needed to find the optimal solution of each algorithm:
  - $H3_0$ (null hypothesis) : The HybridQ algorithm needs more time to find the optimal solution than the other algorithms in the experiments performed.
  - $H3_1$: The HybridQ algorithm does not need more time to find the optimal solution than the other algorithms in the experiments performed.

5.1.4 Experiment phases

The experiment 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.

5.1.5 OpenCart Experiment

The experiment was conducted to test the use of the HybridQ algorithm in a real implemented application. The chosen application was the OpenCart application , available at `opencart.com`. OpenCart is free open source ecommerce platform for online merchants. OpenCart works with PHP 5 and MySQL. The maximum tolerated response time in the test was 5 seconds. The whole process of stress and performance tests, which run for 2 days and with about 1.500 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.

Table 5.2: Q values for response times bellow than service level

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

- Product detail: The application shows details 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.

Q-Learning Training Phase

The application was submitted to 1 hour of training with the Q-learning algorithm using all test scenarios and was obtained the Table 5.2 with the values of q for response times bellow than service level. The action and state with best q-value is increment the number of users ('up') in Add to Cart feature. The learning phase required 1.431 requisitions for application under test.

Results

Figure 5.3 presents the number of requests by maximum fitness value. HybridQ algorithm obtains the maximum value of fitness: 364860 ($H1_1$ hypothesis). HybridQ obtained a solution with greater fitness value, but it needs a much greater number of requests than the other algorithms, not contemplating the hypothesis $H2_1$. GA is the algorithm that obtain the best fitness value with minor number of requests ($H1_0$ hypothesis). All algorithms consume the same time of test (6 hours). The scenario with more fitness value has 4,8 seconds of response time and 38 users:

- 25 users on search page;
- 10 users on Add to Cart feature;

- 2 users removing items from cart;
- 1 users on Main Page.

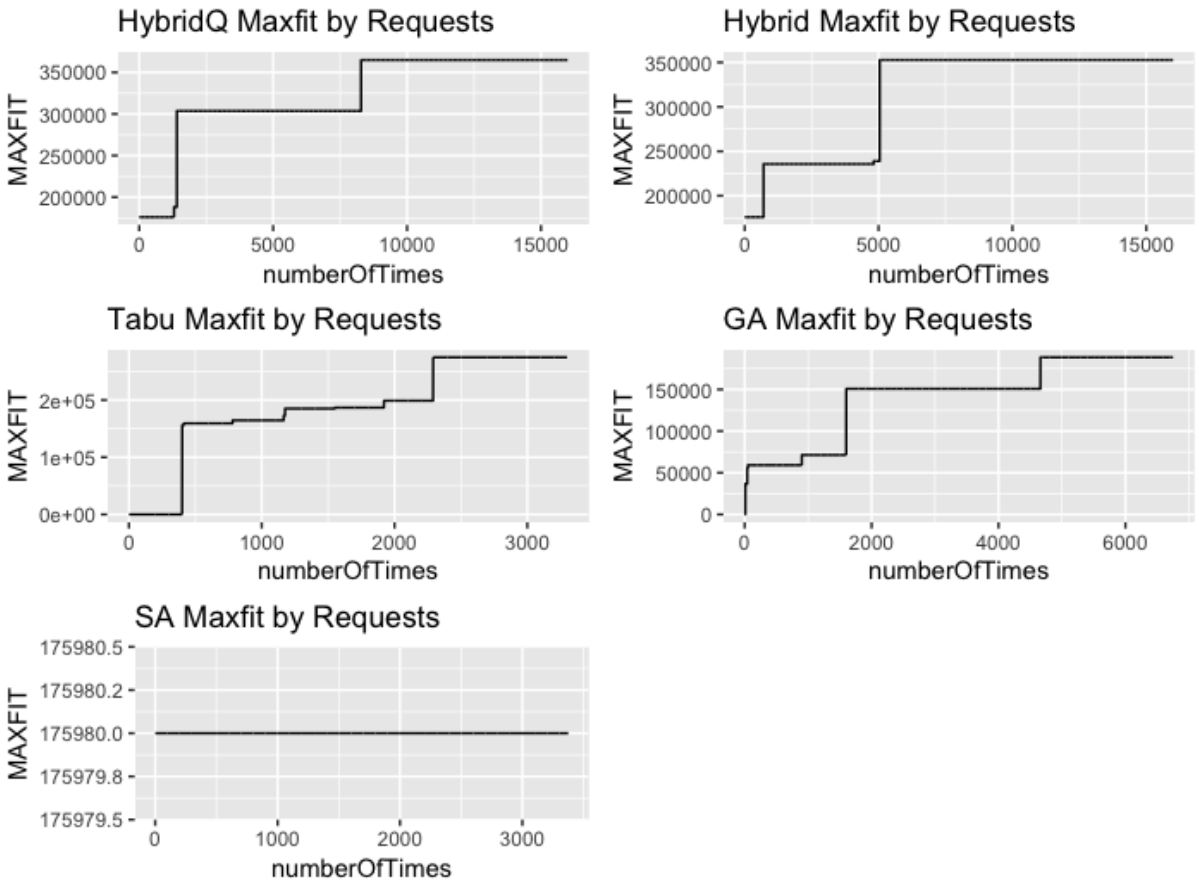


Figure 5.3: Maximum fitness value by number of requests

The t-Test and Wilcoxon Rank Sum Test was applied using the R language. The test results show that HybridQ and HybridQ algorithms is superior than GA, Tabu Search and Simulated Annealing with $p < 0.02$. t-Test shows that the mean of HybridQ fitness value is superior than Hybrid.

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

```
10:  332007.5  313260.9
```

5.1.6 Threats to validity

In this work, we just evaluate the use of single objective algorithm. However, several multiobjective algorithms could be applied. The experiments are performed with the configuration obtained by the irace algorithm, however new experiments are required to verify the sensitivity of the results. It is necessary to compare the current approach with the constraint programming approaches presented in the state of art.

5.2 Conclusion

The present study extends the article "Improving stress search based testing using a hybrid metaheuristic approach" in order to ascertain if the use of the Q-learning technique allows the meta-heuristic algorithms to improve the search for application failures One experiment was conducted to validate the proposed approach. The experiments use genetic algorithms, tabu search, simulated annealing, the hybrid approach and the HybridQ algorithm. The experiment ran for 17 generations. The experiment used an initial population of 5 individuals by meta-heuristics. All tests in the experiment were conducted without the need of a tester, automating the execution of stress tests with the JMeter tool. HybridQ found the individuals with a greater response time. The scenario with greater fitness has 38 users with the Search Page, Add to Cart feature, Removing Item and Main Page features. GA is the algorithm that obtain the best fitness value with minor number of requests. All algorithms consume the same time of test (6 hours).

6 SEARCH-BASED STRESS TESTING USING MULTI-OBJECTIVE HEURISTICS

This chapter present experiments to assert the benefits of multiobjective metaheuristics in search-based stress testing. Multi-objective heuristics may be more suitable for non-functional search-based tests since these tests usually aim to obtain a result with more than one objective, for example, minimizing the number of users of an application, maximize your response time. We examine the use of the multi-objective NSGA-II, SPEA2, PAES and MOEA/D algorithms. The multi-objective algorithms applied in the experiments use an adapted implementation of the jMetal framework (`http://jmetal.sourceforge.net/`). Figure. 6.1 presents the flowchart of Multiobjective Algorithms Adaption. Given an initial population (Figure 6.1 -❶), the multiobjective algorithm implementation receives a set of workloads and generates a new set of individuals (Figure 6.1 -❷). JMeterEngine runs each workload (Figure 6.1 -❸) and the multi-objective algorithm ranks and classifies each workload based on the objective functions (Figure 6.1 -❹). After all these steps the cycle begins until the maximum number of generations is reached.

Several experiments were performed to evaluate the use of multi-objective metaheuristics in search-based stress testing. The experiments use the SEDR adaptation presented in Appendix B. The first experiment uses NSGA-II algorithm to discover application scenarios where there is a high response time for a small number of users. The second experiment presents the experimental results to compare the four multi-objective algorithms. MOEA/D metaheuristics obtained the best hypervolume value when compared with other approaches. Consequently, we propose a collaborative approach using MOEA/D and HybridQ to improve the hypervolume values obtained from the previous experiments. Each algorithm share our best individuals with each other. The fourth experiment presents an experiment to assess the collaborative approach using HybridQ and MOEA/D.

Due to the pervasive and non-deterministic nature of the problem, we do not guarantee that

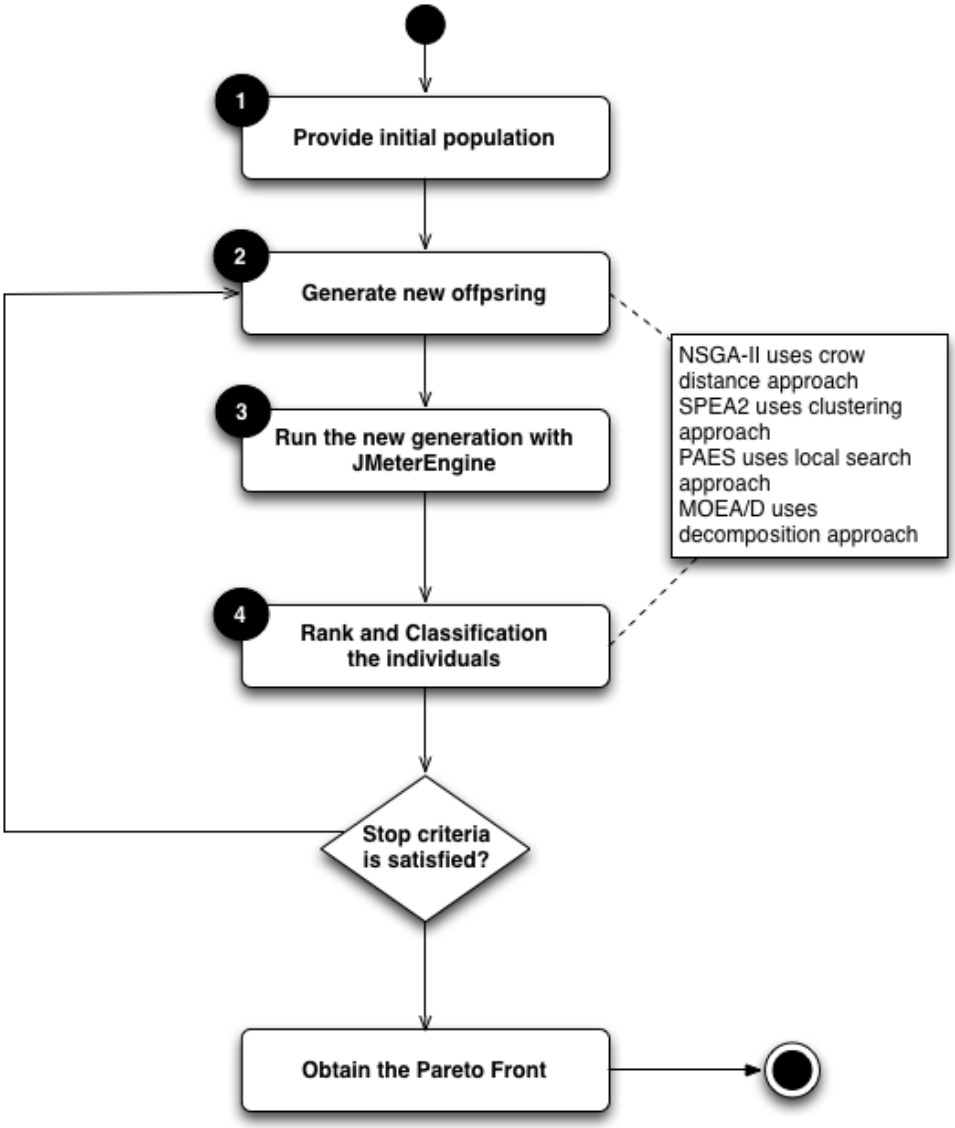


Figure 6.1: Flowchart of implemented algorithms

the same Pareto frontier will be found in different test executions. The fifth experiment presents an initial study to evaluate the Pareto Frontier difference in multiple executions.

6.1 Experiment with multi-object NSGA-II algorithm

In this section, we present the experimental results, in which we carried out to verify the multi-objective NSGA-II implementation. This experiment runs to verify the feasibility of using multi-objective heuristics in search-based stress testing. NSGA-II was initially used for its ease of implementation. This experiment try to discover application scenarios where there is a high response time for a small number of users. The relevance of finding scenarios with superior response times is to enable corrective actions before the application under test is released in a

production environment. 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 milliseconds. The whole process of stress tests, which run for 3 days and 492 executions, was conducted without the need for monitoring by a test designer. The tool automatically selected the next scenarios to be run up to the limit of 123 previously established generations by the algorithm. The experiments use the following application features:

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

The experiments used an initial population of 17 individuals by metaheuristic. The genetic algorithm used the top 10 individuals from each generation in the crossover operation The mutation operation was applied to 10% of the population in each generation. The objective function applied is intended to minimize the number of users and maximize the response time of the scenarios being tested. A penalty applies if an application under test takes a longer time to respond than the expected maximum response time. The experiment used the following objective equations:

$$objectivefunction1 = numberOfUsers - penalty \tag{6.1}$$

$$objectivefunction2 = responsetime - penalty \tag{6.2}$$

The first objective function seeks to find workloads with fewer users. The second objective function seeks to find workloads with longer response times. The penalty is calculated by the following equation:

$$\begin{aligned} penalty &= 100 * \Delta \\ \Delta &= (t_{CurrentResponseTime} - t_{MaximumResponseTimeExpected}) \end{aligned} \tag{6.3}$$

6.1.1 Experiment Research Questions

The following research question is addressed:

- Does the NSGA-II algorithm find relevant workload scenarios according to the two test objectives?

6.1.2 Variables

The independent variable is the use (or not) of NSGA-II algorithm. The dependent variables are the optimal workload scenario found by the algorithm.

6.1.3 Experiment Hypotheses

- With regard to multi-objective algorithms applied in the experiment:
  - H_0 (null hypothesis): The NSGA-II didn't find workloads that meet the two objective functions used in the experiment.
  - H_1: The NSGA-II algorithm found workloads that meet the two objective functions used in the experiment.

6.1.4 Results

Fig. 6.2 and Table 6.1 present the results obtained in the experiment. The experiment found 9 optimal workloads (Pareto Frontier) that present a lower number of users with high response times. Workload number 1 with a single user accessing the dog scenario provided a response time of 245 milliseconds. Workload number 2 with a single user accessing the dog scenario, 7 users accessing the Enter Catalog feature and 3 users in register functionality, provided a response time of 400 seconds.

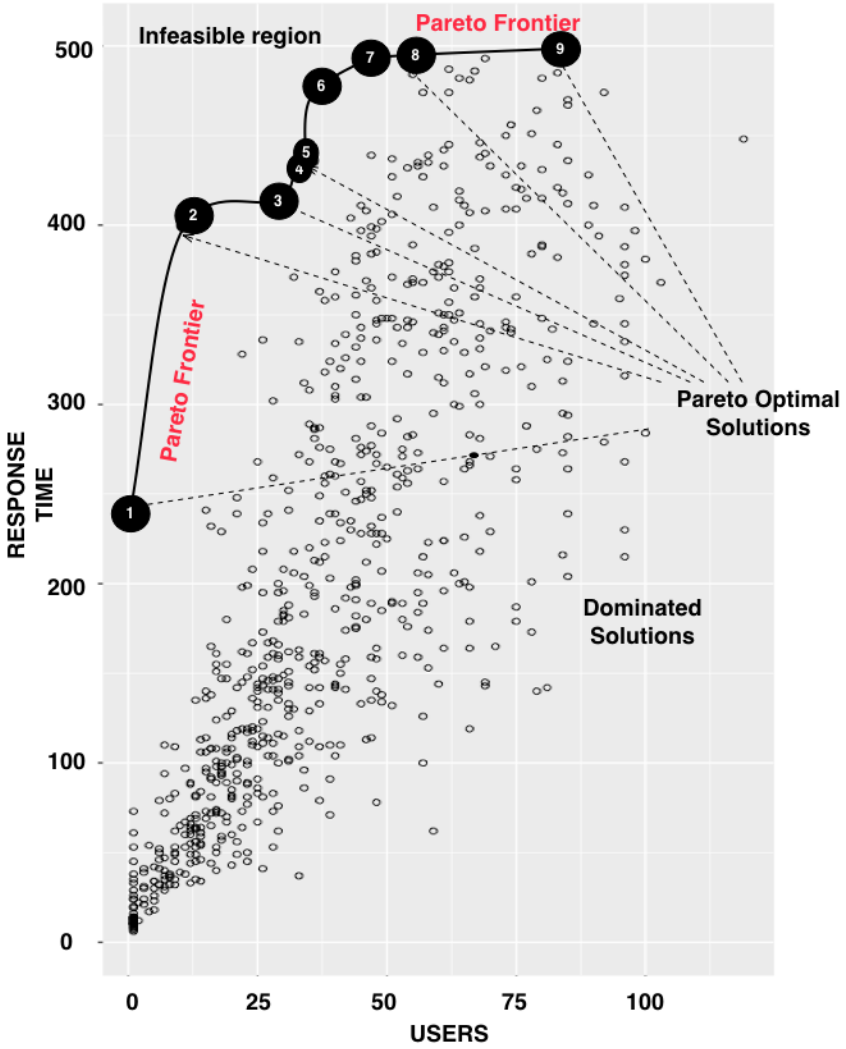


Figure 6.2: Experiment Pareto Frontier

Table 6.1: Experiment Results

N.	USERS	RESP TIME	Dogs Users	Enter Catalog Users	Fish Users	Register Users	Add Rem. Cart	Cart Users
❶	1	245	1					
❷	11	400	1	7		3		
❸	29	416	5	15	4	5		
❹	34	434	16	17				1
❺	35	436	15	4	8	3	5	
❻	37	472	7	13	7	3	7	
❼	47	493	7	11	11	7	7	4
❽	54	496	6		12	8	19	9
❾	85	499		54	12		7	12

6.1.5 Threats to Validity

In this work, we just evaluate the use of one multi-objective algorithm. However, several multi-objective 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.

6.1.6 Experiment Conclusion

The experiment verified the use of a multi-objective algorithm in a search-based stress testing problem. The experiment uses the NSGA-II algorithm to discover application scenarios where there is a high response time for a small number of users. The experiment found 9 optimal workloads that present a lower number of users with high response times. The results of the experiment can help in the decision making of service levels that need to be defined for the application.

6.2 Comparative experiment of Multi-Objective algorithms with Noise Reduction

In this section, we present the experimental results, in which we carried out compare four multi-objective algorithms:

- NSGA-II
- SPEA2
- PAES
- MOEAD

The experiment was undertaken with the JPetStore application with and without a noise reduction strategy. The maximum tolerated response time in the test was 1000 milliseconds. The whole process of stress tests, which run for 5 days and 2226 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 50 previously established generations by the algorithm. The hypervolume metric was used to compare the algorithms. All values used in the hypervolume calculation were normalized. We used the emoa package of the R language to calculate the hypervolume.

6.2.1 Experiment Research Questions

The following research question is addressed:

- Which, among the four algorithms chosen, is the most suitable multi objective algorithm for the search-based test problem?

6.2.2 Variables

The independent variable is the algorithms: NSGA-II, SPEA2, PAES and MOEA/D. The dependent variables are the hypervolume of each algorithm.

6.2.3 Experiment Hypotheses

- With regard to the relevance of test results:
  - H_0 (null hypothesis): The multi-objective with the higher value of hypervolume don't find the scenarios with higher response time and lower number of users.
  - H_1: The multi-objective with the higher value of hypervolume find the scenarios with higher response time and lower number of users.

- With regard to hypervolume of multi-objective algorithms applied in the experiment:
  - H_1: MOEA/D is the algorithm with the higher value of hypervolume in the experiments.
  - H_2: NSGA-II is the algorithm with higher value of hypervolume in the experiments.
  - H_3: SPEA2 is the algorithm with the higher value of hypervolume in the experiments.
  - H_4: PAES is the algorithm with the higher value of hypervolume in the experiments.

6.2.4 Experiment Results

Table 6.2 shows the hypervolume value of each algorithm. For the hypervolume calculus, the values of the number of users and response time were transformed to the same scale. The highest hypervolume algorithm was the MOEA/D. The test scenarios present in the Pareto frontier were:

Table 6.2: Hypervolume by algorithm with Noise Reduction

ALGORITHM	HYPERVOLUME	DIMENSIONALITY	POINT DENSITY
MOEA/D	10.185576	2	7817.328935
NSGA-II	7.541011	2	7726.815663
PAES	7.240581	2	7758.493105
SPEA2	5.539566	2	7772.269274

- 1 user on Enter the Catalog scenario and response time of 18 milliseconds;
- 5 users and response time of 35 milliseconds (1 user on Dog scenario, 1 user on Register scenario, 1 user on Fish scenario and 1 user on Cart scenario);
- 22 users and response time of 788 milliseconds (3 users on AddRemoveCart scenario, 7 users on Fish scenario, 5 users on Cart scenario and 7 users on Register scenario);
- 72 users and response time of 1074 milliseconds (14 users on Fish scenario, 11 users on Register scenario, 11 users on Cart scenario, 19 users on Enter the Catalog scenario and 17 users on Dogs scenario)
- 89 users and response time of 1396 (8 users on Dogs scenario, 35 users on Cart scenario, 29 users on AddRemoveCart scenario, 7 users on Enter the Catalog scenario and 10 users on Fish scenario)

MOEA/D found two scenarios that approached the established service level for the experiment of 1000 milliseconds. NSGA-II found a scenario with 49 users and response time of 685 and a scenario with 59 users and response time of 1138 milliseconds. Although the MOEA/D algorithm found the scenario with the lowest number of users with a time close to the service level, the NSGA-II found a scenario with more users with a response time closer to the service level. Figures 6.3, 6.4, 6.5 and 6.6 present the Pareto frontier for each algorithm.

6.2.5 Experiment Conclusion

The multi-objective with the higher value of hypervolume find the scenarios with higher response time and lower number of users. However, each algorithm presented relevant results for stress testing. MOEA/D was the algorithm with the highest hypervolume value.

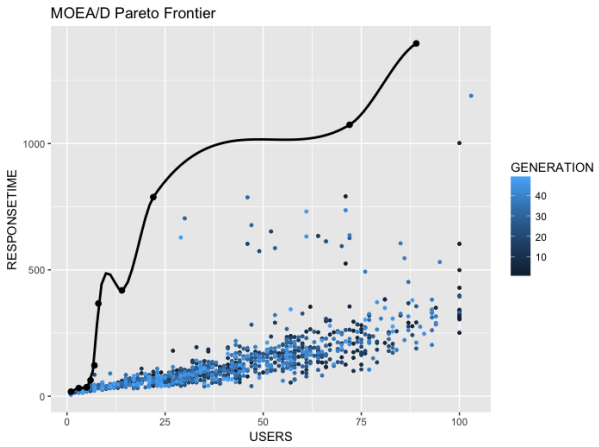


Figure 6.3: MOEA/D Pareto Frontier

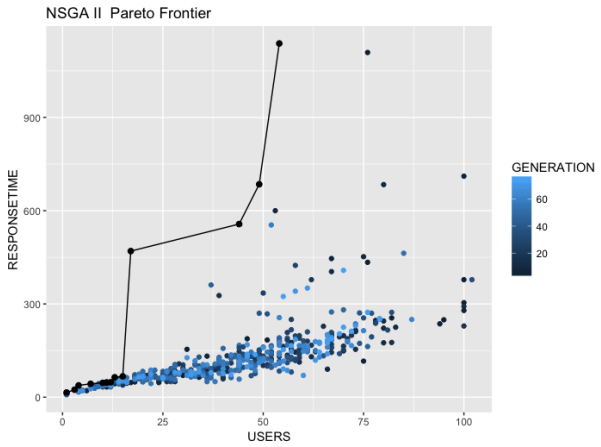


Figure 6.4: NSGA II Pareto Frontier

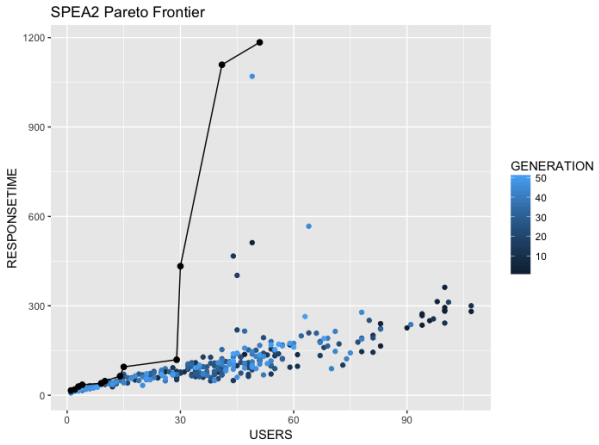


Figure 6.5: SPEA2 Pareto Frontier

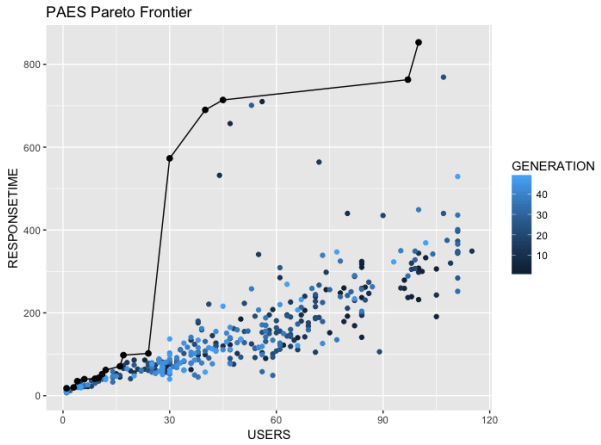


Figure 6.6: PAES Pareto Frontier

6.2.6 Threats to Validity

Due to the non-deterministic nature of the problem, we do not guarantee that the same Pareto frontier will be found in different test executions. The workloads found by the MOEA/D algorithm do not dominate all workloads found by all others algorithms.

6.3 Comparative Experiment between HybridQ and MOEA/D

This section presents the experimental results carried out to compare the MOEA/D multi-objective with the HybridQ algorithm. The experiment was conducted with JPetStore application with the SEDR adapted noise reduction strategy. The maximum tolerated response time in the test was 1000 milliseconds. The whole process of stress tests, which run for 1 day and 534 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 15 previously established

generations by the algorithm.

6.3.1 Experiment Research Questions

The following research question is addressed:

- The solutions presented by a single objective algorithm are dominated by all solutions presented by a multi-objective algorithm?

6.3.2 Variables

The independent variable is algorithms: HybridQ and MOEA/D. The dependent variables are the Pareto frontier points found by each algorithm.

6.3.3 Experiment Hypotheses

- With regard to multi-objective and single objective algorithms applied in the experiment:
  - H_0 (null hypothesis): All solutions found by HybridQ are not dominated by the MOEA/D algorithm.
  - H_1: All solutions found by HybridQ are dominated by the MOEA/D algorithm.

6.3.4 Experiment Results

Fig. 6.8 presents the result of the experiment. Most of the individuals found by the HybridQ algorithm are dominated by those found by MOEA/D. The exception is a single maximum point found by HybridQ. From the results, a hybrid collaborative approach was proposed, where the results between the MOEA / D and HybridQ algorithms would be shared.

6.4 Experiment with HybridQ and MOEA/D Collaborative approach

This section presents an experiment to evaluate the collaborative approach using HybridQ and MOEA/D. The proposition consists of each algorithm share our best individuals with each other. The experiment was conducted with JPetStore application with the SEDR adapted noise reduction strategy. The maximum tolerated response time in the test was 1000 milliseconds.

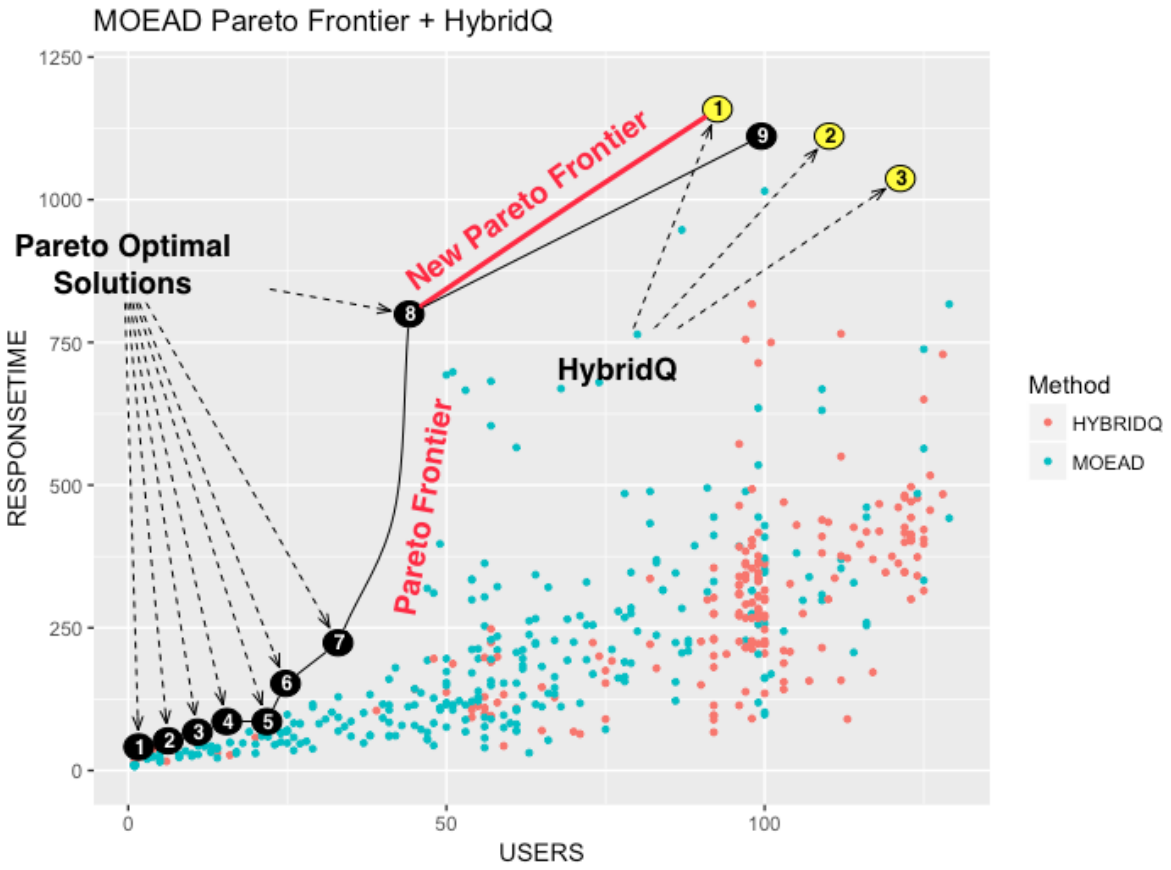


Figure 6.7: HybridQ and MOEA/D Pareto Frontier

The whole process of stress tests, which run for 1 day and 983 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 50 previously established generations by the algorithm.

6.4.1 Experiment Research Questions

The following research question is addressed:

- The combined use of MOEA/D with Hybrid could improve the results obtained in the previous experiments with MOEA/D?

6.4.2 Variables

The independent variable is the collaborative approach of the algorithms HybridQ and MOEA/D. The dependent variable is the hypervolume found by the new approach.

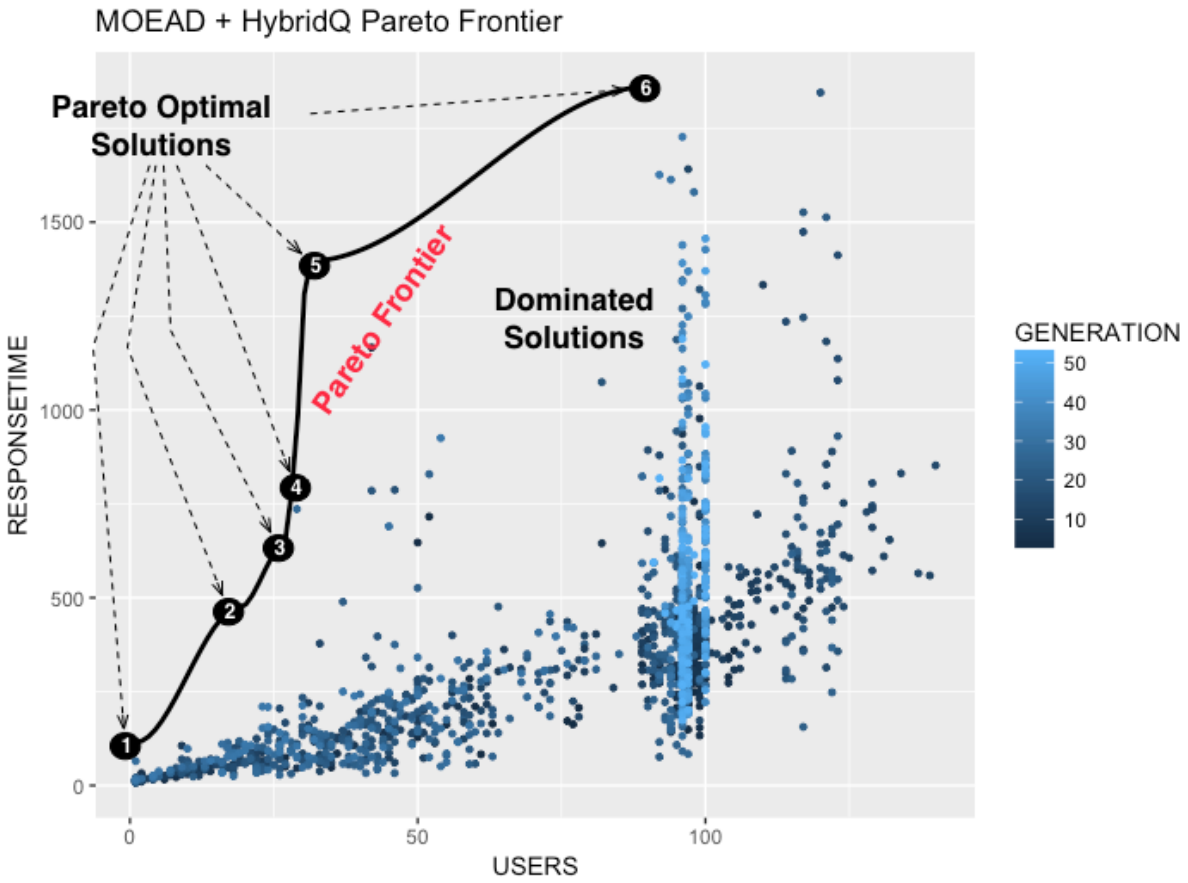


Figure 6.8: Collaborative approach Pareto Frontier

6.4.3 Experiment Hypotheses

- With regard to hypervolume:
  - H_0 (null hypothesis): The hypervolume of the new approach don't surpass the previous experiments.
  - H_1: The hypervolume of the new approach surpass the previous experiments.

6.4.4 Experiment Results

At the end of 28 generations, the new approach obtains a hypervolume value of the 10.29. The new approach obtains a hypervolume of 10.84 at the end of 50 generations. Table 6.3 and 6.8 presents 6 workloads that which are the Pareto optimal solution in the experiment. The algorithm found a workload with 31 users and response time of 1396, above the expected response time of 1000 milliseconds.

6.4.5 Experiment Conclusion

The results presented by the MOEA/D with HybridQ surpass previous approaches for the same number of generations.

Table 6.3: Experiment Results

N.	USERS	RESP TIME	Dogs Users	Enter Catalog Users	Fish Users	Register Users	Add Rem. Cart	Cart Users
❶	1	115					1	
❷	19	473	6		5	4	4	
❸	27	644	17		3			7
❹	28	774		8	3	3	8	6
❺	31	1396	6			13	4	8
❻	90	1860	12	11	3	3	30	31

6.5 Analysis of Pareto Frontier in Multiple Executions

This section provides an initial study to evaluate the Pareto frontier difference in multiple executions. The experiment was conducted with the JPetStore application with the SEDR adapted noise reduction strategy. The maximum tolerated response time in the test was 1000 milliseconds. The whole process of stress tests, which run for 2 days and 983 executions. The tool automatically selected the next scenarios to be run up to the limit of 23 previously established generations by the algorithm. Three executions were carried out with different initial populations. Figs. 6.9 and 6.10 presents the comparative results between the three executions (1, 2 and 3).

6.5.1 Experiment Research Questions

The following research question is addressed:

- There is an acceptable variation of the Pareto frontier for the three test executions?

6.5.2 Variables

The independent variable is three test executions using the collaborative approach with the algorithms HybridQ and MOEA/D. The dependent variable are: the hypervolume and Pareto frontier found by each test execution.

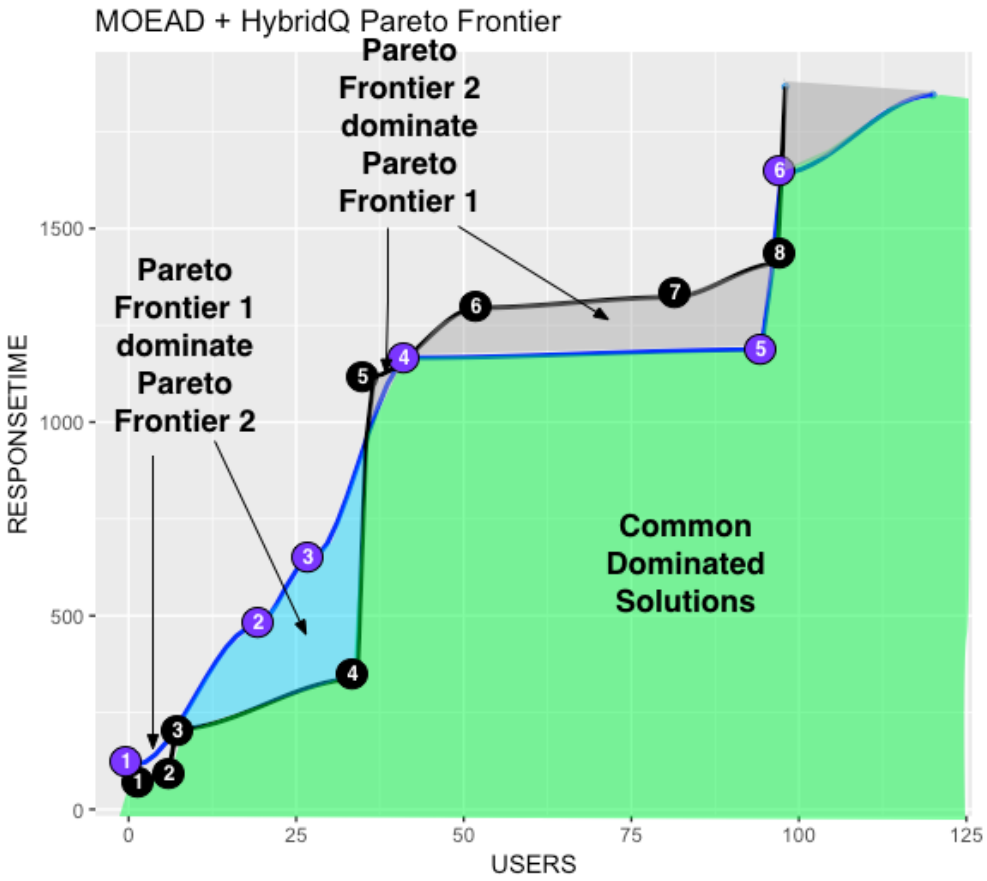


Figure 6.9: Pareto Frontier of the first and second test executions

Table 6.4: Experiment Results

Test Execution	Hypervolume		
1	9.377189	**Average**	8.652338
2	9.648418	**Standard Deviation**	1.49652728764998
3	6.931407	**Percent deviation**	0.172962185209359

6.5.3 Experiment Hypotheses

- H_0 (null hypothesis): The hypervolume variation of the three tests performed is greater than 10%.

- H_1: The hypervolume variation of the three tests performed is less than 10%.

6.5.4 Experiment Results

Table 6.3 presents the experiment results. The variation between stayed around 17%. Further studies are needed to ascertain more accurately the variation of the Pareto frontier.

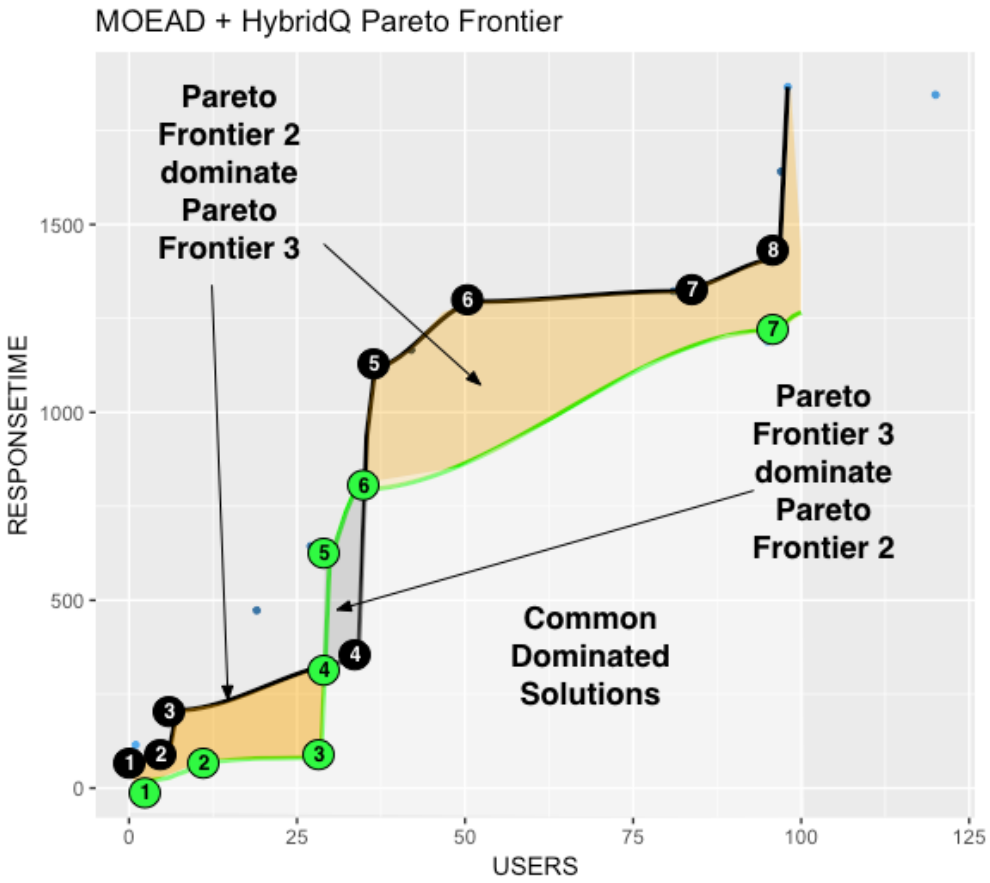


Figure 6.10: Pareto Frontier of the second and third test executions

6.6 A Survey Study of use of Multi-Objective Algorithms in Search-based Stress testing

We conducted a survey study to evaluate the use of multi-objective algorithm by the non-functional test team of the Serpro, a company of the Brazilian Federal Government in Fortaleza. The team consists of 4 test designers from 36 to 39 years old. A test designer has 12 years of stress testing experience and 3 test designers have from 1 to 5 years of testing experience.

The experiment took place in July of this year and consists of two stages. The first stage consisted of using the IAdapter tool by test designers. In the second stage, the designers answered a set of questions to evaluate the applied techniques and the IAdapter tool. The initial proposal was to evaluate the use of the JMeter tool with and without the IAdapter plugin. However, the time available for the experiment was reduced due to the participation of test designers in some company projects, allowing the application of the experiment only with the IAdapter plugin. All the test designers who participated in the experiment already had previous experience of using the Apache JMeter tool.

At the beginning of the experiment, test designers had brief guidelines for using IAdapter and the technique of search-based stress testing. After the initial orientations, designers generated the initial population and started the tests. The experiment used the HybridQ and MOEA/D algorithms. The questionnaires had 3 objective questions using the Likert scale and two open questions related to the strengths and weaknesses of the applied techniques. The first question asks if there is an inherent difficulty in the activity of stress testing to find the best or worst case scenarios.

Table 6.5: Survey of use of Multi-Objective Algorithms - Question 1

How much do you agree with the statement below?	
There is an inherent difficulty in the activity of stress testing to find the best or worst case scenarios	
Response Item	**Number of answers**
Strongly Agree	4
Agree	0
Neither	0
Disagree	0
Strongly Disagree	0

The second question asks if there is an improvement in the use of search-based stress testing. Most test designers agreed that the use of search-based tests can improve the team's current test practice using an approach based on capturing and replay test scripts.

Table 6.6: Survey of use of Multi-Objective Algorithms - Question 2

How much do you agree with the statement below?	
The use of search-based tests could improve the practice of stress testing	
Response Item	**Number of answers**
Strongly Agree	0
Agree	3
Neither	1
Disagree	0
Strongly Disagree	0

The third question asks if the IAdapter tool found an important test usually not found by the testers.

Among the positive points cited by the test designers are:

- Automatic identification of new test scenarios.
- Identification of the best or worst test scenario.

Table 6.7: Survey of use of Multi-Objective Algorithms - Question 3

How much do you agree with the statement below?	
The IAdapter tool found an important test not usually found by the testers	
Response Item	**Number of answers**
Strongly Agree	0
Agree	3
Neither	0
Disagree	1
Strongly Disagree	0

- Ease of Use, which allowed test designers to concentrate more on the analysis of results rather than the execution and design of the tests.
- The tool allows you to quickly configure multiple test batteries.
- Ability to use learning algorithms.
- The tool enables the implementation of new algorithms, making it available according to JMeter standards.

Among the negative points cited by the test designers are:

- There is still no way to turn off the verbosity of the plugin's log.
- Cannot use other databases besides MySQL.
- Absence of a parameter for setting the minimum number of users.
- Graphics need to be a bit more intuitive.

6.7 Conclusion

The use of multi-objective metaheuristics has made it possible to delimit a frontier where the response times are below the previously established service level. MOEA/D metaheuristics obtained the best hypervolume value when compared with NSGA II, SPEA2 and PAES. The MOEA / D algorithm dominates most of the results presented by HybridQ. However, we have verified from the experiments that some of the results found by the HybridQ algorithm can contribute to the use of MOEA/D. Thus, we propose the collaborative use of MOEA/D and HybridQ algorithms. The collaborative approach improves the hypervolume values obtained and found more relevant workloads than the previous experiments.

For the time being, we cannot state that the benefits found by using MOEA / D and HybridQ algorithms together can be obtained in other contexts. Due to the pervasive and non-deterministic nature of the problem, we do not guarantee that the same Pareto frontier will be found in different test executions. More experiments are needed to verify changes related to the Pareto frontier in different executions.

7 CONCLUSION

This chapter presents our conclusions, revisits the contributions, shows some limitations, and proposals for future work. In this thesis, we deal with the use of hybrid metaheuristics, reinforcement learning and multi-objective metaheuristics in Search-based stress testing. A tool named IAdapter, a JMeter plugin for performing search-based load tests, was developed. Several experiments were conducted to validate the proposed approach.

7.1 Summary of contributions and Achievements

First, we proposed a hybrid collaborative algorithm to perform search-based stress testing. In order to enhance the efficiency of our proposal, we utilize the reinforcement learning technique with the aforementioned hybrid algorithm. We also proposed the use of multi-objective algorithms in search-based stress testing. To this end, we propose an adaptation of the SEDR algorithm for noise reduction and a new approach to the use of a multiobjective metaheuristic in conjunction with the HybridQ algorithm. We conduct numerous experiments to evaluate the proposed approaches. The use of multi-objective metaheuristics has made it possible to delimit a frontier where the response times are below the previously established service level. Table 7.1 presents the thesis Research Question by metaheuristic. The subsequent subsections present the summary of results for each research question.

7.1.1 (Research Question 1) How to improve search-based stress testing using single-objective metaheuristics?

The hybrid approach uses genetic algorithms, tabu search, and simulated annealing. Three experiments were performed to validate the hybrid metaheuristic. The first experiment aimed to execute stress testing on a simulated component. In the first experiment, the signed-rank Wilcoxon non-parametrical procedure was used for comparing the results. The procedure showed that there was a significant improvement in results with the Hybrid Metaheuristic ap-

proach when compared with the other algorithms. All tests in the experiment were conducted without the need of a tester, automating stress testing with the JMeter tool (GOIS et al., 2016).

7.1.2 (Research Question 2) How to improve the choice of neighboring solutions in a single-objective metaheuristic to explore, reducing the time needed to obtain the scenarios with the longest response time in the application?

The HybridQ algorithm is used 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 that HybridQ is more expensive than Hybrid because of Q-learning. The research has as the premise that the same application under performance tests can be submitted to more than one cycle of tests execution, decreasing the cost of the exploration phase of the Q-learning algorithm used. HybridQ algorithm obtains the maximum value of fitness, but it needs a much greater number of requests than the other algorithms. GA was the algorithm that obtains the best fitness value with a minor number of requests.

7.1.3 (Research Question 3) How to use multi-objective metaheuristics in search-based stress testing to obtain a Pareto frontier to improve the definition of SLAs?

The experiment with NSGA-II algorithm discover application scenarios where there is a high response time for a small number of users. The experiment found 8 optimal workloads that present a lower number of users with high response times. The results of the experiment can help in the decision making of service levels that need to be defined for the application (GOIS; PORFIRIO; COELHO, 2017). MOEA/D metaheuristics obtained the best hypervolume value when compared with NSGA II, SPEA2 and PAES. The MOEA/D algorithm dominates most of the results presented by HybridQ. However, we have verified from the experiments that some of the results found by the HybridQ algorithm can contribute to the use of MOEA/D. The collaborative approach (HybridQ+MOEA/D) improves the hypervolume values obtained and found more relevant workloads than the previous experiments. The experiments found scenarios that contributes to the definition of the service level. To take an example, the second experiment with the hybrid algorithm found scenarios with only 35 users but which already have a response time greater than the maximum defined for the service level.

Table 7.1: Thesis Research Questions

Metaheuristic	Research Question
Hybrid	**(Research Question 1)** How to improve search-based stress testing using single-objective metaheuristics?
Hybrid + Reinforcement Learning	**(Research Question 2)** How to improve the choice of neighboring solutions in a single-objective metaheuristic to explore, reducing the time needed to obtain the scenarios with the longest response time in the application?
Multi-objective + Hybrid + Reinforcement + Learning	**(Research Question 3)** How to use multi-objective metaheuristics in search-based stress testing to obtain a Pareto frontier to improve the definition of SLAs?

7.2 Open Issues

There is a range of future improvements in the proposed approach. Also as a typical search strategy, it is sometimes 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 necessary to have an adequate termination criterion to stop the search process.

For the time being, we cannot state that the benefits found by using MOEA/D and HybridQ algorithms together can be obtained in other contexts. Due to the pervasive and non-deterministic nature of the problem, we do not guarantee that the same Pareto frontier will be found in different test executions. More experiments are needed in order to verify changes related to the Pareto frontier in different executions.

The choice of fitness function can be a difficult step in Genetic Algorithms. Different fitness functions promote different GA behavior. The fitness function is still manually adjusted and additional studies need to be performed to get the best weight values for the fitness function. There is a great difficulty in comparing some approaches present in the state-of-art, due to the lack of availability of the tools used. The present research compared the proposed approach with other methods based on the use of single or multiobjective metaheuristics.

7.3 Future Works

Among the future works of the research, the use of different combinatorial optimization algorithms such as very large-scale neighborhood search is one that we can highlight. Further research should ascertain more exhaustively the variation of the Pareto frontier. Further exper-

iments with reinforcement learning in conjunction with multi-objective search algorithms must be performed. The PAES algorithm performs a local search is can be benefited with the strategies outlined in this research. Additional experiments are needed to see if the benefits found by this thesis in the context of stress testing will be obtained in other contexts.

BIBLIOGRAPHY

AFZAL, W.; TORKAR, R.; FELDT, R. A systematic review of search-based testing for non-functional system properties. In: . [S.l.]: Elsevier B.V., 2009. v. 51, n. 6, p. 957–976. ISBN 0950-5849. ISSN 09505849.

ALANDER, J. T. J.; MANTERE, T.; TURUNEN, P. Genetic Algorithm Based Software Testing. In: *Neural Nets and Genetic Algorithms*. [S.l.: s.n.], 1998.

ALESIO, S. D. et al. Worst-Case Scheduling of Software Tasks – A Constraint Optimization Model to Support Performance Testing. In: . [S.l.: s.n.], 2014. p. 813–830.

ALESIO, S. D.; SEN, S. Using UML/MARTE to support performance tuning and stress testing in real-time systems. *Software and Systems Modeling*, Springer Berlin Heidelberg, 2017. ISSN 1619-1366. Disponível em: <http://link.springer.com/10.1007/s10270-017-0585-x>.

ALESIO, S. D. I. et al. Combining Genetic Algorithms and Constraint Programming. In: . [S.l.: s.n.], 2015. v. 25, n. 1.

ALETI, A.; MOSER, I.; GRUNSKE, L. Analysing the fitness landscape of search-based software testing problems. *Automated Software Engineering*, p. 1–19, 2016. ISSN 15737535.

ANAND, S. et al. An orchestrated survey of methodologies for automated software test case generation. In: . [S.l.: s.n.], 2013. v. 86, p. 1978–2001. ISSN 01641212.

ARAIZA-ILLAN, D.; PIPE, A. G.; EDER, K. Model-based Test Generation for Robotic Software: Automata versus Belief-Desire-Intention Agents. p. 1–16, 2016. Disponível em: <http://arxiv.org/abs/1609.08439>.

ARANTES, A. O. et al. Tool support for generating model-based test cases via web. *International Journal of Web Engineering and Technology*, v. 9, n. 1, p. 62–96, 2014. ISSN 14761289. Disponível em: <http://dx.doi.org/10.1504/IJWET.2014.063041>.

ARCAINI, P.; GARGANTINI, A.; RICCOBENE, E. Improving Model-based Test Generation by Model Decomposition. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, p. 119–130, 2015. Disponível em: <http://doi.acm.org/10.1145/2786805.2786837>.

ARCELLI, D.; CORTELLESSA, V.; TRUBIANI, C. Antipattern-Based Model Refactoring for Software Performance Improvement. In: . [s.n.], 2012. p. 33–42. ISBN 9781450313469. Disponível em: <http://doi.acm.org/10.1145/2304696.2304704>.

ARSLAN, M. et al. Automatic performance analysis of cloud based load testing of web-application & its comparison with traditional load testing. *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, v. 2015-Novem, n. September, p. 140–144, 2015. ISSN

23270594. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84958250651&partnerID=tZOtx3y1>.

AVRITZER, A.; LARSON, B. Load Testing Software Using Deterministic State Testing. In: . New York, NY, USA: ACM, 1993. (ISSTA '93), p. 82–88. ISBN 0-89791-608-5. Disponível em: <http://doi.acm.org/10.1145/154183.154244>.

AVRITZER, A.; WEYUKER, E. Generating test suites for software load testing. In: . New York, New York, USA: ACM Press, 1994. p. 44–57. ISBN 0897916832.

AVRITZER, A.; WEYUKER, E. The automatic generation of load test suites and the assessment of the resulting software. *IEEE Transactions on Software Engineering*, v. 21, n. 9, p. 705–716, 1995. ISSN 0098-5589. Disponível em: <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=464549&url=http%3A%2F%2Fieeexplore.ieee.org%2F http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=464549&url=http://ieeexplore.ieee.org/xpls/abs_all.jsp?a

BAI, X. et al. Cloud testing tools. In: *Proceedings - 6th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2011*. [S.l.: s.n.], 2011. p. 1–12. ISBN 9781467304108.

BARBER, S. User Community Modeling Language (UCML ™) v1 . 1 for Performance Test Workloads UCML ™ Overview. In: . [S.l.: s.n.], 1999. p. 1–9.

BARROS, M. D.; SHIAU, J. Web services wind tunnel: On performance testing large-scale stateful web services. In: . [S.l.: s.n.], 2007.

BAYAN, M.; CANGUSSU, J. W. Automatic feedback, control-based, stress and load testing. *Proceedings of the 2008 ACM symposium on Applied computing - SAC '08*, p. 661, 2008. ISSN 1063-6773. Disponível em: <http://portal.acm.org/citation.cfm?doid=1363686.1363847>.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys*, v. 35, n. 3, p. 189–213, 2003. ISSN 02545330.

BRIAND, L. C.; LABICHE, Y.; SHOUSHA, M. Stress testing real-time systems with genetic algorithms. In: . [S.l.: s.n.], 2005. p. 1021. ISBN 1595930108.

BROWN, W. H. et al. *AntiPatterns: refactoring software, architectures, and projects in crisis*. [S.l.]: John Wiley & Sons, Inc., 1998.

BUCHS, D.; LUCIO, L.; CHEN, A. Model checking techniques for test generation from business process models. *Reliable Software Technologies–Ada-Europe 2009*, p. 59–74, 2009.

CAI, Y.; GRUNDY, J.; HOSKING, J. Synthesizing client load models for performance engineering via web crawling. *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering - ASE '07*, p. 353, 2007. Disponível em: <http://portal.acm.org/citation.cfm?doid=1321631.1321684>.

CANFORA, G. et al. An approach for QoS-aware service composition based on genetic algorithms. In: . [S.l.: s.n.], 2005. ISBN 1595930108.

CORPORATION, M. *Performance Testing Guidance for Web Applications*. United States?: Microsoft Press, nov. 2007. 288 p. Disponível em: <http://www.amazon.com/Performance-Testing-Guidance-Web-Applications/dp/0735625700 http://msdn.microsoft.com/en-us/library/bb924375.aspx>.

CORTELLESSA, V.; FRITTELLA, L. A Framework for Automated Generation of Architectural Feedback from Software Performance Analysis. In: . [S.l.: s.n.], 2007. p. 171–185.

DEAN, L.; DON, W. *Managing Software Requirements: A Use Case Approach*. [S.l.]: Addison Wesley, 2003.

DEB, K. *Multi-objective optimization using evolutionary algorithms*. [S.l.]: John Wiley & Sons, 2001.

DEB, K. et al. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *Parallel Problem Solving from Nature PPSN VI*, p. 849–858, 2000. ISSN 10871357. Disponível em: <http://repository.ias.ac.in/83498/>.

DEB, K.; MOHAN, M.; MISHRA, S. Evaluating the epislon-domination based multiobjective evolutionary algorithm for a quick computation of Pareto-optimal solutions. *Evolutionary Computation Journal*, v. 13, n. 4, p. 501–525, 2005.

Di Alesio, S. et al. Stress testing of task deadlines: A constraint programming approach. In: . [S.l.: s.n.], 2013. p. 158–167. ISBN 9781479923663.

Di Lucca, G. a.; FASOLINO, A. R. Testing Web-based applications: The state of the art and future trends. In: . [S.l.: s.n.], 2006. v. 48, p. 1172–1186. ISBN 0-7695-2413-3. ISSN 09505849.

Di Pietro, A.; WHILE, L.; BARONE, L. Applying evolutionary algorithms to problems with noisy, time-consuming fitness functions. *Proceedings of the 2004 Congress on Evolutionary Computation*, p. 1254–1261, 2004. Disponível em: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1331041>.

DRAHEIM, D. et al. Realistic load testing of Web applications. In: *Conference on Software Maintenance and Reengineering (CSMR'06)*. [S.l.: s.n.], 2006. ISBN 0-7695-2536-9. ISSN 1052-8725.

DURILLO, J. J. et al. jmetal: a java framework for developing multi-objective optimization metaheuristics. *Science And Technology*, v. 01, n. March 2016, p. 1–12, 2006.

DUSTIN, E.; RASHKA, J.; PAUL, J. *Automated Software Testing: Introduction, Management, and Performance*. [S.l.: s.n.], 1999. 575 p. ISBN 0201432870.

ENOIU, E. P.; SUNDMARK, D.; PETTERSSON, P. Model-based test suite generation for function block diagrams using the UPPAAL model checker. *Proceedings - IEEE 6th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2013*, p. 158–167, 2013.

ERINLE, B. *Performance Testing With JMeter 2.9*. [S.l.: s.n.], 2013. ISBN 9781782165842.

FANG, L. et al. Formal model-based test for AUTOSAR multicore RTOS. *Proceedings - IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012*, p. 251–259, 2012. ISSN 2159-4848.

FEITELSON, D. G. *Workload Modeling for Computer Systems Performance Evaluation*. [S.l.]: Cambridge University Press, 2013.

FIEBRINK, R.; MCKAY, C.; FUJINAGA, I. Combining d2k and jgap for efficient feature weighting for classification tasks in music information retrieval. *Ismir*, p. 510–513, 2005. Disponível em: <ismir2005.ismir.net/proceedings/2121.pdf?>.

GANESAN, D. et al. Experience Report: Model-Based Test Automation of a Concurrent Flight Software Bus. *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, p. 445–454, 2016. ISSN 10719458.

GAROUSI, V. Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models using Genetic Algorithms. In: . [S.l.: s.n.], 2006. ISBN 9780494262252.

GAROUSI, V. Empirical analysis of a genetic algorithm-based stress test technique. In: . [S.l.: s.n.], 2008. p. 1743. ISBN 9781605581309.

GAROUSI, V. A Genetic Algorithm-Based Stress Test Requirements Generator Tool and Its Empirical Evaluation. In: . [S.l.: s.n.], 2010. v. 36, n. 6, p. 778–797. ISSN 0098-5589.

GAY, G.; RAYADURGAM, S.; HEIMDAHL, M. P. Automated Steering of Model-Based Test Oracles to Admit Real Program Behaviors. *IEEE Transactions on Software Engineering*, PP, n. 99, 2016. ISSN 00985589.

Gendreau, Michel and Potvin, J.-Y. *Handbook of Metaheuristics*. [S.l.: s.n.], 2010. ISBN 9781441979605.

GIESE, H.; GRAF, J.; WIRTZ, G. Seamless visual object-oriented behavior modeling for distributed software systems. *Visual Languages, 1999. Proceedings. 1999 IEEE Symposium on*, p. 156–199, 1999.

GLOVER, F.; MARTÍ, R. Tabu Search. In: . [S.l.: s.n.], 1986. p. 1–16.

GOIS, N.; PORFIRIO, P.; COELHO, A. A multi-objective metaheuristic approach to search-based stress testing. In: *Proceedings of the 2017 IEEE International Conference on Computer and Information Technology (CIT)*. [S.l.: s.n.], 2017.

GOIS, N. et al. Improving Stress Search Based Testing using a Hybrid Metaheuristic Approach. In: *Proceedings of the 2016 Latin American Computing Conference (CLEI)*. [S.l.: s.n.], 2016. p. 718–728. ISBN 978-1-5090-1632-7.

GONçALVES, M. C. Um Processo de Inferência de Desempenho para Apoiar o Planejamento da Capacidade de Aplicações na Nuvem Um Processo de Inferência de Desempenho para Apoiar o Planejamento da Capacidade de Aplicações na Nuvem. In: . [S.l.: s.n.], 2014.

GRECHANIK, M.; FU, C.; XIE, Q. Automatically finding performance problems with feedback-directed learning software testing. In: . [S.l.]: Ieee, 2012. p. 156–166. ISBN 978-1-4673-1067-3.

GREENWALD, A.; HALL, K.; SERRANO, R. Correlated Q-learning. In: . [s.n.], 2003. p. 84–89. Disponível em: <http://www.aaai.org/Papers/Symposia/Spring/2002/SS-02-02/SS02-02-012.pdf>.

GROSS, H.; JONES, B. F.; EYRES, D. E. Structural performance measure of evolutionary testing applied to worst-case timing of real-time systems. In: . [S.l.: s.n.], 2000. v. 147, n. 2, p. 25–30. ISBN 0818669101. ISSN 14625970.

HALILI, E. H. *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites.* [S.l.: s.n.], 2008. ISSN 1098-6596. ISBN 9788578110796.

HARMAN, M.; JIA, Y.; ZHANG, Y. Achievements , open problems and challenges for search based software testing. *8th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, n. Icst, 2015. Disponível em: <http://www0.cs.ucl.ac.uk/staff/mharman/icst15.pdf>.

HARMAN, M.; MCMINN, P. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. In: . [S.l.: s.n.], 2010. v. 36, n. 2, p. 226–247. ISBN 0098-5589. ISSN 00985589.

HESSEL, A. *Model-Based Test Case Generation for Real-Time Systems*. [S.l.: s.n.], 2007. ISBN 9789155468835.

HIERONS, R. M. et al. Using formal specifications to support testing. In: . [S.l.: s.n.], 2009. v. 41, n. 2, p. 1–76. ISSN 0360-0300.

HONG, T.-P.; WANG, H.-S.; CHEN, W.-C. Simultaneously applying multiple mutation operators in genetic algorithms. In: . [S.l.]: Springer, 2000. v. 6, n. 4, p. 439–455.

ILLES, T. et al. Criteria for Software Testing Tool Evaluation – A Task Oriented View. v. 2, p. 213–222, 2005.

J. Wegener, K. Grimm, M. Grochtmann, H. Sthamer, B. J. Systematic testing of real-time systems. In: . [S.l.: s.n.], 1996.

JANSSENS, G.; PANGILINAN, J. Multiple criteria performance analysis of non-dominated sets obtained by multi-objective evolutionary algorithms for optimisation. *Artificial Intelligence Applications and Innovations*, Springer, p. 94–103, 2010.

JEONG, S.-y. et al. State Transition Based Test Model and Test Case Generation Technique for Embedded System: An Empirical Approach. v. 10, n. 11, p. 233–254, 2016.

JIANG, Z. *Automated analysis of load testing results*. Tese (Doutorado), 2010. Disponível em: <http://dl.acm.org/citation.cfm?id=1831726>.

KIM, G. B. A method of generating massive virtual clients and model-based performance test. *Proceedings - International Conference on Quality Software*, v. 2005, p. 250–254, 2005. ISSN 15506002.

KIRAN, S.; MOHAPATRA, A.; SWAMY, R. Experiences in performance testing of web applications with Unified Authentication platform using Jmeter. *2nd International Symposium on Technology Management and Emerging Technologies, ISTMET 2015 - Proceeding*, p. 74–78, 2015.

KITCHENHAM, B.; CHARTERS, S. Guidelines for performing Systematic Literature Reviews in Software Engineering. *Engineering*, v. 2, p. 1051, 2007. ISSN 00010782.

KNOWLES, J.; CORNE, D. The Pareto archived evolution strategy: a new baseline algorithm\nfor Pareto multiobjective optimisation. *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, v. 1, p. 98–105, 1999. ISSN 1879-1026.

LACOUR, R.; KLAMROTH, K.; FONSECA, C. M. A box decomposition algorithm to compute the hypervolume indicator. *Computers and Operations Research*, p. 1–21, 2015. ISSN 03050548.

LENZ, C.; CHIMIAK-OPOKA, J.; BREU, R. Model-Driven Testing of SOA-based Software. ... *of the SEMSOA Workshop on Software* ..., 2007. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.128.7626&rep=rep1&type=pdf>.

LEWIS, W. E.; DOBBS, D.; VEERAPILLAI, G. *Software testing and continuous quality improvement*. [s.n.], 2005. 688 p. ISBN 1420080733. Disponível em: <http://books.google.com/books?id=fgaBDd0TfT8C&pgis=1>.

LUO, Q. et al. FOREPOST: finding performance problems automatically with feedback-directed learning software testing. In: . [S.l.: s.n.], 2015. p. 1–51. ISBN 1066401594. ISSN 15737616.

Manuel López-Ibãnez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, T. S. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, v. 3, p. 43–58, 2016. ISSN 22147160.

MARINESCU, R. et al. *A Research Overview of Tool-Supported Model-based Testing of Requirements-based Designs*. [S.l.: s.n.], 2015. 89–140 p. ISSN 00652458. ISBN 9780128021323.

Mark Utting, A. P.; LEGEARD, B. A taxonomy of model-based testing approaches. In: . [S.l.: s.n.], 2012. v. 24, n. 8, p. 297–312. ISBN 1099-1689. ISSN 10991689.

Matnei Filho, R. A.; VERGILIO, S. R. A multi-objective test data generation approach for mutation testing of feature models. *Journal of Software Engineering Research and Development*, Journal of Software Engineering Research and Development, v. 4, n. 1, p. 4, 2016. ISSN 2195-1721. Disponível em: <http://jserd.springeropen.com/articles/10.1186/s40411-016-0030-9>.

Matthias Beyer, Winfried Dulz, K.-S. H. Performance Issues in Statistical Testing. n. April 2006, 2014.

MCCONAGHY, T. et al. Trustworthy genetic programming-based synthesis of analog circuit topologies using hierarchical domain-specific building blocks. *IEEE Transactions on Evolutionary Computation*, v. 15, n. 4, p. 557–570, 2011. ISSN 1089778X.

MENASCÉ, D. A. Load Testing, Benchmarking, and Application Performance Management for the Web. *Int. CMG Conference*, n. January 2002, p. 271–282, 2002.

MENASCÉ, D. A.; MASON, G. TPC-W : A Benchmark for E-commerce. In: . [S.l.: s.n.], 2002. p. 1–6.

MICHALAK, K. The effects of asymmetric neighborhood assignment in the MOEA/D algorithm. *Applied Soft Computing Journal*, Elsevier B.V., v. 25, p. 97–106, 2014. ISSN 15684946. Disponível em: <http://dx.doi.org/10.1016/j.asoc.2014.07.029>.

MODEL-BASED generation of testbeds for web services. In: . [S.l.: s.n.], 2008. p. 266–282. ISBN 978-3-540-68514-2, 978-3-540-68524-1. ISSN 978-3-540-68514-2.

Mohammad S. Obaidat, P. N.; ZARAI, F. *Modeling and Simulation of Computer Networks and Systems Methodologies and Applications*. [S.l.: s.n.]. ISBN 9780128008874.

MOLYNEAUX, I. *The Art of Application Performance Testing: Help for Programmers and Quality Assurance*. 1st. ed. [S.l.]: "O'Reilly Media, Inc.", 2009. 159 p. ISBN 9780596551056.

MOSCHER, M.; FÖGEN, K. Facing synthetic workload generation as part of performance testing–a tools approach. *Full-scale Software Engineering/The Art of Software Testing*, p. 38, 2017.

MUELLER, F.; WEGENER, J. A comparison of static analysis and evolutionary testing for the verification of timing constraints. In: . [S.l.: s.n.], 1998. ISBN 0-8186-8569-7.

NACHIYAPPAN, S.; JUSTUS, S. Cloud Testing Tools and Its Challenges : A Comparative Study. *Procedia - Procedia Computer Science*, Elsevier Masson SAS, v. 50, p. 482–489, 2015. ISSN 1877-0509. Disponível em: <http://dx.doi.org/10.1016/j.procs.2015.04.018>.

OLTEAN, M.; ABRAHAM, A.; MARIO, K. Multiobjective Optimization Using Adaptive Pareto Archived Evolution Strategy. p. 1–6, 2005.

PARASURAMAN, A.; BERRY, L. L.; ZEITHAML, V. A. Understanding customer expectations of service. *Sloan Management Review*, v. 32, n. 3, p. 39–48, 1991.

PENTA, M. D.; CANFORA, G.; ESPOSITO, G. Search-based testing of service level agreements. In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. [S.l.: s.n.], 2007. p. 1090–1097. ISBN 9781595936974.

PERRY, W. E. *Effective methods for software testing*. [s.n.], 2004. ISSN 14337851. ISBN 9780764598371. Disponível em: <http://onlinelibrary.wiley.com/doi/10.1002/cbdv.200490137/abstract\nhttp://scholar.google.com/scholar?l

POHLHEIM, H.; CONRAD, M.; GRIEP, A. Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences. In: . [S.l.: s.n.], 2005. p. 804—–814.

PUCHINGER, J.; RAIDL, R. Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization : A Survey and Classification. In: . [S.l.: s.n.], 2005. v. 3562, p. 41–53. ISBN 9783540263197. ISSN 03029743.

PUSCHNER, P.; NOSSAL, R. Testing the results of static worst-case execution-time analysis. In: . [S.l.: s.n.], 1998. ISBN 0-8186-9212-X. ISSN 1052-8725.

RAIDL, G. R.; PUCHINGER, J.; BLUM, C. Metaheuristic hybrids. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2010. p. 469–496.

RAIDL, R. A Unified View on Hybrid Metaheuristics. In: . [S.l.: s.n.], 2006. p. 1–12. ISBN 9783540463849. ISSN 03029743.

RAJESHWARI, B. S. Service Level Agreement based Scheduling Techniques in Cloud : A Survey Service Level Agreement based Scheduling Techniques in Cloud : A Survey. n. January, 2016.

RAKSHIT, P.; KONAR, A.; DAS, S. Noisy evolutionary optimization algorithms – A comprehensive survey. *Swarm and Evolutionary Computation*, Elsevier, v. 33, p. 18–45, 2017. ISSN 22106502. Disponível em: <http://dx.doi.org/10.1016/j.swevo.2016.09.002>.

RAUF, I.; IQBAL, M. Z. Z.; MALIK, Z. I. Model Based Testing of Web Service Composition Using UML Profile. *2nd Workshop on Model-based Testing in Practice, MOTIP 2009*, 2009. Disponível em: <http://www.fokus.fraunhofer.de/go/motip09>.

RODRIGUES, E. M. et al. Evaluating Capture and Replay and Model-based Performance Testing Tools: An Empirical Comparison. *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, p. 9:1—-9:8, 2014. ISSN 19493789. Disponível em: <http://doi.acm.org/10.1145/2652524.2652587>.

SAEED, A.; Ab Hamid, S. H.; SANI, A. A. Cost and Effectiveness of Search-Based Techniques for Model-Based Testing: An Empirical Analysis. *International Journal of Software Engineering and Knowledge Engineering*, v. 27, n. 04, p. 601–622, 2017. ISSN 0218-1940. Disponível em: <http://www.worldscientific.com/doi/abs/10.1142/S021819401750022X>.

San Miguel, J. L.; TAKADA, S. GUI and Usage Model-based Test Case Generation for Android Applications with Change Analysis. *Proceedings of the 1st International Workshop on Mobile Development*, p. 43–44, 2016. Disponível em: <http://doi.acm.org/10.1145/3001854.3001865>.

SANDLER, C.; BADGETT, T.; THOMAS, T. The Art of Software Testing. In: . [S.l.]: John Wiley & Sons, 2004. p. 200. ISBN 9781118133156.

SCHAEFER, C.; DO, H.; SLATOR, B. M. Crushinator: A framework towards game-independent testing. *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013 - Proceedings*, p. 726–729, 2013.

SHOUSHA, M. *Performance Stress Testing of Real-Time Systems Using Genetic Algorithms*. Tese (Doutorado) — Carleton University Ottawa, 2003.

SIEGMUND, F.; NG, A. H. C.; DEB, K. A comparative study of dynamic resampling strategies for guided Evolutionary Multi-objective Optimization. *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, n. 2013008, p. 1826–1835, 2013.

SILVEIRA, M. da; RODRIGUES, E.; ZORZO, A. Generation of Scripts for Performance Testing Based on UML Models. In: . [S.l.: s.n.], 2011.

SMITH, C.; WILLIAMS, L. Software Performance AntiPatterns; Common Performance Problems and their Solutions. In: . [s.n.], 2002. v. 2, p. 797–806. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.6968&rep=rep1&type=pdf>.

SMITH, C. U.; WILLIAMS, L. G. Software performance antipatterns. In: . [s.n.], 2000. p. 127–136. ISBN 158113195X. Disponível em: <http://portal.acm.org/citation.cfm?doid=350391.350420>.

SMITH, C. U.; WILLIAMS, L. G. More New Software Performance AntiPatterns: EvenMore Ways to Shoot Yourself in the Foot. In: . [s.n.], 2003. p. 717–725. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.4517&rep=rep1&type=pdf>.

SRIDHAR, A.; SRINIVASULU, D.; MOHAPATRA, D. P. Model-based test-case generation for Simulink/Stateflow using dependency graph approach. *Proceedings of the 2013 3rd IEEE International Advance Computing Conference, IACC 2013*, p. 1414–1419, 2013.

SULLIVAN, M. O. et al. Testing Temporal Correctness of Real-Time Systems — A New Approach Using Genetic Algorithms and Cluster Analysis —. In: . [S.l.: s.n.], 1998. p. 1–20.

SUTTON, R. S.; BARTO, A. G. Reinforcement learning. In: . [S.l.: s.n.], 2012. v. 3, n. 9, p. 322. ISBN 0262193981. ISSN 18726240.

TALBI, E.-G. *Metaheuristics: from design to implementation*. [S.l.]: John Wiley & Sons, 2009.

TALBI, E.-G. *Metaheuristics: From Design to Implementation*. [S.l.: s.n.], 2013. 1689–1699 p. ISSN 1098-6596. ISBN 9788578110796.

TERVONEN, T.; KINGDOM, U. Evaluation of multi-objective optimization approaches for solving green supply chain design problems : Evaluation of multi-objective optimization approaches for solving green supply chain design problems. n. April, 2017.

TRACEY, N. J. *A search-based automated test-data generation framework for safety-critical software*. Tese (Doutorado) — Citeseer, 2000.

TRACEY, N. J.; CLARK, J. a.; MANDER, K. C. Automated Programme Flaw Finding using Simulated Annealing. In: . [S.l.: s.n.], 1998.

TRENT, G.; SAKE, M. WebSTONE: The first generation in {HTTP} server benchmarking. In: . [S.l.: s.n.], 1995.

TRUBIANI, C. Automated generation of architectural feedback from software performance analysis results Catia Trubiani. *Language*, 2011.

TRUBIANI, C. PhD Thesis in Computer Science Automated generation of architectural feedback from software performance analysis results Catia Trubiani. *Language*, 2011.

UTTING, M.; LEGEARD, B. *Practical model-based testing: a tools approach*. [S.l.]: Morgan Kaufmann, 2010.

VOGELE, C. et al. WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction???a model-driven approach for session-based application systems. In: . Springer Berlin Heidelberg, 2016. p. 1–35. ISSN 16191374. Disponível em: <"http://dx.doi.org/10.1007/s10270-016-0566-5>.

WANG, X.; ZHOU, B.; LI, W. Model-based load testing of web applications. In: . [s.n.], 2013. v. 36, n. 1, p. 74–86. ISSN 0253-3839. Disponível em: <http://www.tandfonline.com/doi/abs/10.1080/02533839.2012.726028>.

WEGENER, J.; GROCHTMANN, M. Verifying timing constraints of real-time systems by means of evolutionary testing. In: . [S.l.: s.n.], 1998. v. 15, n. 3, p. 275–298. ISBN 0922-6443. ISSN 0922-6443.

WEGENER, J. et al. Testing real-time systems using genetic algorithms. In: . [s.n.], 1997. v. 6, n. 2, p. 127–135. ISSN 0963-9314, 1573-1367. Disponível em: <http://www.springerlink.com/index/uh26067rt3516765.pdf>.

Wegener, Joachim and Pitschinetz, Roman and Sthamer, H. Automated Testing of Real-Time Tasks. In: . [S.l.: s.n.], 2000.

WERT, A.; HAPPE, J.; HAPPE, L. Supporting swift reaction: Automatically uncovering performance problems by systematic experiments. In: . [S.l.: s.n.], 2013. p. 552–561. ISBN 9781467330763. ISSN 02705257.

WERT, A. et al. Automatic detection of performance anti-patterns in inter-component communications. In: . [s.n.], 2014. p. 3–12. ISBN 9781450325776. Disponível em: <http://dx.doi.org/10.1145/2602576.2602579>.

WIECZOREK, S.; STEFANESCU, A.; ROTH, A. Model-Driven Service Integration Testing - A Case Study. *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, p. 292–297, 2010.

WILLIAMS, L. G.; SMITH, C. U. PASASM : A Method for the Performance Assessment of Software Architectures. *Proceedings of the third international workshop on Software and performance - WOSP '02*, n. January 2002, p. 179, 2002. Disponível em: <http://portal.acm.org/citation.cfm?doid=584369.584397>.

WOEHRLE, M. Search-based stress testing of wireless network protocol stacks. *Proceedings - IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012*, p. 794–803, 2012. ISSN 2159-4848.

Xinying Cai, H. Z. Model-based Test Generation for Software Product Line. 2007.

YE, L. Model-Based Testing Approach for Web Applications. n. June, 2007.

YOO, S.; HARMAN, M. Pareto efficient multi-objective test case selection. *Proceedings of the 2007 international symposium on Software testing and analysis - ISSTA '07*, p. 140, 2007. Disponível em: <http://doi.acm.org/10.1145/1273463.1273483%5Cnhttp://dl.acm.org/ft_gateway.cfm?id=1273483&type=pdf%

YUEN, T. J.; RAMLI, R. Comparison of Computational Efficiency of Moea \ D and Nsga-Ii for Passive Vehicle Suspension Optimization. v. 2, n. Cd, 2009.

ZHANG, Q.; LI, H. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation*, v. 11, n. 6, p. 712–731, 2007. ISSN 1089-778X.

ZITZLER, E.; LAUMANNS, M.; THIELE, L. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, p. 95–100, 2001. ISSN 03772217.

ZITZLER, E.; THIELE, L. Multiobjective evolutionary algorithms: a comparative case study\nand the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, v. 3, n. 4, p. 257–271, 1999. ISSN 1089-778X.

APPENDIX A – STRESS TESTING

This chapter surveys the state of the art literature in stress testing research. The thesis extends the survey presented by Jiang et al. (JIANG, 2010) and Afzal et al. (AFZAL; TORKAR; FELDT, 2009) to the Stress Testing context. This survey will be helpful for stress testing practitioners and software engineering researchers with interests in testing and analyzing software systems. The paper uses the systematic review method proposed by Kitchenham (KITCHENHAM; CHARTERS, 2007). The systematic review is based on a comprehensive set of 97 articles obtained after a multi-stage selection process and have been published in the time span 1994–2016.

A.1 Research Questions

In order to examine the evidence of stress testing properties, we proposed the following two research questions:

- How modeling a stress test workload?
- What are the main anti-patterns found by stress tests?

A.1.1 Study Selection Criteria

The population in this study is the domain of software testing. Intervention includes the application of stress test techniques to test different types of non-functional properties. The primary studies used in this review were obtained from searching databases of peer-reviewed software engineering research that met the following criteria: contains peer-reviewed software engineering journals articles, conference proceedings, and book chapters; contains multiple journals and conference proceedings, which include volumes that range from 1996 to 2017.

The resulting list of databases was:

•ACM Digital Library

•Google Scholar

•IEEE Electronic Library

•Inspec

•Scirus (Elsevier)

•SpringerLink

The search strategy was based on the following steps: Identification of alternate words and synonyms for terms used in the research questions. This is done to minimize the effect of differences in terminologies. Identify common stress testing properties for searching. Use of Boolean OR to join alternate words and synonyms and use of Boolean AND to join major terms.

We used the following search terms:

•Load Testing: load test, Load Testing

•Stress Testing: stress test, stress testing

•Performance Testing: Performance tests

•Test tools: JMeter, load runner, performance tester

The idealized selection process was done in two parts: an initial document selection of the results that could reasonably satisfy the selection criteria based on a title and the articles abstract reading, followed by a final selection of the initially selected papers based on the introduction and conclusion reading of the papers. The following exclusion criteria are applicable in this review, i.e. exclude studies that:

•Do not relate to stress testing.

•Do not relate to load testing tool.

•Do not relate to load/stress testing model.

From 366 initial papers, 97 papers were selected.

A.2 Stress Test Process

Contrary to functional testing, which has clear testing objectives, Stress testing objectives are not clear in the early development stages and are often defined later on a case-by-case basis. The Fig. A.1 shows a common Load, Performance and Stress test process (JIANG, 2010).

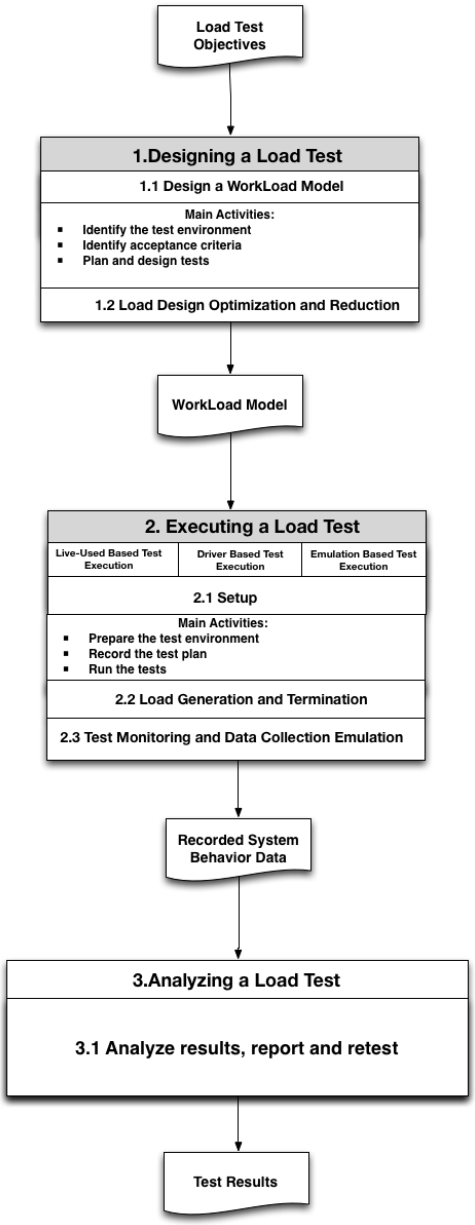


Figure A.1: Load, Performance and Stress Test Process (JIANG, 2010)(ERINLE, 2013)

The goal of the load design phase is to devise a load, which can uncover non-functional problems. Once the load is defined, the system under test executes the load and the system behavior under load are recorded. Load testing practitioners then analyze the system behavior to detect problems (JIANG, 2010).

Once a proper load is designed, a load test is executed. The load test execution phase consists of the following three main aspects: (1) Setup, which includes system deployment and test execution setup; (2) Load Generation and Termination, which consists of generating the load; and (3) Test Monitoring and Data Collection, which includes recording the system behavior during execution(JIANG, 2010).

The core activities in conducting a usual Load, Performance and Stress tests are (ERINLE, 2013):

- Identify the test environment: identify test and production environments and knowing the hardware, software, and network configurations help derive an effective test plan and identify testing challenges from the outset.
- Identify acceptance criteria: identify the response time, throughput, and resource utilization goals and constraints.
- Plan and design tests: identify the test scenarios.In the context of testing, a scenario is a sequence of steps in an application. It can represent a use case or a business function such as searching a product catalog, adding an item to a shopping cart, or placing an order (CORPORATION, 2007). This task includes a description of the speed, availability, data volume throughput rate, response time, and recovery time of various functions, stress, and so on. This serves as a basis for understanding the level of performance and stress testing that may be required to each test scenario (LEWIS; DOBBS; VEERAPILLAI, 2005).
- Prepare the test environment: configure the test environment, tools, and resources necessary to conduct the planned test scenarios.
- Record the test plan: record the planned test scenarios using a testing tool.
- Run the tests: Once recorded, execute the test plans under light load and verify the correctness of the test scripts and output results.
- Analyze results, report, and retest: examine the results of each successive run and identify areas of bottleneck that need addressing.

Utting et al. presents four classic test process (UTTING; LEGEARD, 2010):

- Manual Testing process;
- Capture/Replay Testing process;

•Script-based Testing process;

•Keyword-Driven Automated Testing process;

In a Manual Testing process, the execution is done manually for each test case. The tester follows a test case and interacts directly with the application under test. The manual-designing of the tests is time-consuming and not ensure systematic coverage of the application. Capture/Replay attempts to reduce the cost of test re-execution by capturing interactions with the application during a test execution session and replay those interactions in later test executions (UTTING; LEGEARD, 2010).

A test script is an executable script that runs one or more test cases. The Script-based testing process tries to resolve the test execution problem by automating it. Keyword-driven testing, or action-word testing, takes this a step further by using action keywords in the test cases, in addition to data. Each action keyword corresponds to a fragment of a test script (UTTING; LEGEARD, 2010).

A.3 Stress Test Execution

The stress test execution consists of deploying the system and setup test execution; generating workloads according to the configurations and terminating the load when the load test is completed and recording the system behavior. There are three general approaches of load test executions (MOLYNEAUX, 2009)(JIANG, 2010):

•Live-User Based Executions: The test examines a system's behavior when the system is simultaneously used by many users or execute a load test by employing a group of human testers.

•Driver Based Executions: The driver based execution approach automatically generate thousands or millions of concurrent requests for a long period of time using a software tool.

•Emulation Based Executions: The emulation based load test execution approach performs the load testing on special platforms and doesn't require a fully functional system and conduct load testing.

Usually, a stress test execution it is performed with Driver Based Executions approaches (ERINLE, 2013) (Mohammad S. Obaidat; ZARAI,) (WANG; ZHOU; LI, 2013). There are three categories of load drivers (JIANG, 2010):

- Benchmark Suite: specialized load driver, designed for one type of system. For example, LoadGen is a load driver specified used to load test the Microsoft Exchange MailServer.

- Centralized Load Drivers: refer to a single load driver, which generates the load.

- Peer-to-peer Load Drivers: refer to a set of load drivers, which collectively generate the target testing load. Peer-to-peer load drivers usually have a controller component, which coordinates the load generation among the peer load drivers.

A stress test needs to perform hundreds or thousands of concurrent requests to the application under test. Automated tools are needed to carry out serious load, stress, and performance testing. Sometimes, there is simply no practical way to provide reliable, repeatable performance tests without using some form of automation. The aim of any automated test tool is to simplify the testing process.

A.3.1 Stress Testing Tools

There are several tools to the execution of stress testing. Stress testing tools are software products based on workload models to generate request sequences similar to real requests. They are designed and implemented as versatile software tools for performing tuning or capacity planning studies. Usually, the tool functions are semi-automated, whereas the execution of the tests itself is performed by a tool, the choice of scenarios to be executed as well as the decision to start new execution batteries are activities of the test designer or tester. Normally, load test tools use test scripts. Test scripts are written in a GUI testing framework or a back-end server-directed performance tool such as JMeter. These frameworks are the basis on which performance testing is mostly done in industry. Performance test scripts imitate large numbers of users to create a significant load on the application under tests. Stress testing tools typically have the following components (GRECHANIK; FU; XIE, 2012) (MOLYNEAUX, 2009):

- Scripting module: Enable recording of end-user activities in different middleware protocols;

- Test management module: Allows the creation of test scenarios;

- Load injectors: Generate the load with multiple workstations or servers;

- Analysis module Provides the ability to analyze the data collected by each test iteration.

Comparing stress test tools is a laborious and difficult task since they offer a large amount and diversity of features (DUSTIN; RASHKA; PAUL, 1999). In next subsection we present studies that contrast stress testing tools according to a wide set of features and capabilities, focusing on their ability to realize search-based tests or have learning capacities. In the following subsection, we present details about the JMeter tool and the reasons why it was chosen as the object of the present research.The stress test tools were categorized in three different groups (Mohammad S. Obaidat; ZARAI,):

- Benchmarks that model the client and server paradigm in the Web context.
- Software products to evaluate performance and functionality of a given Web application, such as LoadRunner, WebLOAD, and JMeter.
- Testing tools and other approaches for traffic generation based on HTTP traces.

A.3.2 Comparative Studies

Illes et al. present a systematic approach for evaluation criteria for test tools. Using the TORE methodology the study identify activities which potentially could be automated or at least supported by a test tool. The study evaluates three tools: WinRunner, Rational Robot and HTTrace. WinRunner is distributed by Mercury. an. HTTrace is not a commercial product but developed for internal use for the company i-TV-T. The study evaluates the tools in the tasks: TestPlanning and monitoring, designing test cases, constructing test cases, executing test cases and analyzing test cases. Key features of all three test tools are the construction of test cases by capturing and subsequently editing the test scripts and the execution of the recorded test scripts. WinRunner and Rational Robot can be extended to provide test planning and monitoring as well as defect and reporting facilities. HTTrace's strength lies on testing database applications by allowing the reset of consistent database states. Additionally, all three tools can be extended to provide support for testing quality attributes of the system under test (ILLES et al., 2005).

Tables A.1 and A.2 summarizes the studied workloads generators as well as the grade (full or partial) in which they fulfill the features described below. None of the presented tools uses heuristic or learning resources when choosing the scenarios to be tested or the workloads to be applied in the test.

Penã-Ortiz et al. present a study where stress test tools are compared using 12 features (Mohammad S. Obaidat; ZARAI,):

- Distributed architecture. This refers to the ability to distribute the generation process among different nodes.
- Analytical-based architecture. This feature represents the capability to use analytical and mathematical models to define the workload.
- Business-based architecture. When defining a testing environment, the simulator architecture should implement the same features as the real environment.
- Client parameterization. This is the ability to parameterize generator nodes.
- Workload types. Some generators organize the workload in categories or types.
- Testing the Web application functionality (functional testing).
- Multi-platform refers to a software package that is implemented in multiple types of computer platforms, inter-operating among them.
- Differences between LAN and WAN.
- The generator should be a friendly application.
- The load test tool has performance reports.
- The load test tool is open-source.
- Users' dynamism. The users have the ability of change the behavior in during the test.

A.3.3 Benchmarks Group

WebStone was designed by Silicon Graphics in 1996 to measure the performance of Web server software and hardware products. Nowadays, both executable and source actualized code for WebStone are available for free. The benchmark generates a Web server load by simulating multiple Web clients navigating a website. All the testing done by the benchmark is controlled by a Webmaster, which is a program that can be run on one of the client computers or on a different one (Mohammad S. Obaidat; ZARAI,) (TRENT; SAKE, 1995) .

TPC Benchmark (TPC-W) is a transitional Web benchmark defined by the Transaction Processing Performance Council that models a representative e-commerce evaluating the architecture performance on a generic profile.The models use a remote browser emulator to generate requests to the server under test. TPC-W adopts the CBMG model to define the workloads in

Table A.1: Benchmarks group

Feature/Tool	WebStone	SpecWeb	SURGE	Web Polygraph	TPC-W
Analytical-Based Architecture	Full sup.	Full sup.	Full sup.	Full sup.	Full sup.
Distributed Architecture	Full sup.	Full sup.	Full sup.	Full sup.	
Business-Based Architecture		Partial Support		Partial Support	Full sup.
Client Parameterization	Full sup.	Full sup.		Partial Support	Full sup.
Workload Types		Full sup.			Full sup.
Functional Testing					
LAN and WAN					
Multi-platform	Full sup.	Full sup.	Full sup.	Full sup.	Full sup.
Ease of Use					
Performance Reports	Partial Support	Full sup.		Full sup.	Full sup.
Open Source	Full sup.		Full sup.	Partial Support	Full sup.
User's Dynamism					Partial Support

spite of this model only characterizing user dynamic behavior partially. The remote browser emulators are located on the client side and generate workload towards the e-commerce Web application, which is located on the server side (e-commerce server) (Mohammad S. Obaidat; ZARAI,) (MENASCÉ; MASON, 2002).

Open STA is an open source software developed in C++ and released under the GPL license. OpenSTA provides a scripting language which permits to simulate the activity of a user. This language can describe HTTP/S scenario and all the test executions is managed in a graphical interface. The composition of the test is very simple, allowing the tester choose scripts for a test and a remote computer that will execute each test.

A.3.4 Software Products

LoadRunner is one of the most popular industry-standard software products for functional and performance testing. It was originally developed by Mercury Interactive, but nowadays it is commercialized by Hewlett-Packard. LoadRunner supports the definition of user navigation, which is represented using a scripting language. The basic steps are recorded, creating a shell script. Next, this script is then taken off-line and undergoes further manual steps such as data parameterization and correlations. Finally, the desired performance scripts are obtained after

Table A.2: Software products

Feature/Tool	LoadRunner	WebLOAD	JMeter
Analytical-Based Architecture	Partial Support	Partial Support	Partial Support
Distributed Architecture	Full support	Full support	Full support
Business-Based Architecture	Full support	Full support	Full support
Client Parameterization	Full support	Full support	Full support
Workload Types	Full support	Full support	Partial Support
Functional Testing	Full support	Full support	Partial Support
LAN and WAN			
Multi-platform	Full support	Full support	Full support
Ease of Use	Full support	Full support	Partial Support
Performance Reports	Partial Support	Full support	Full support
Open Source			Partial Support
User's Dynamism	Partial Support	Partial Support	Partial Support

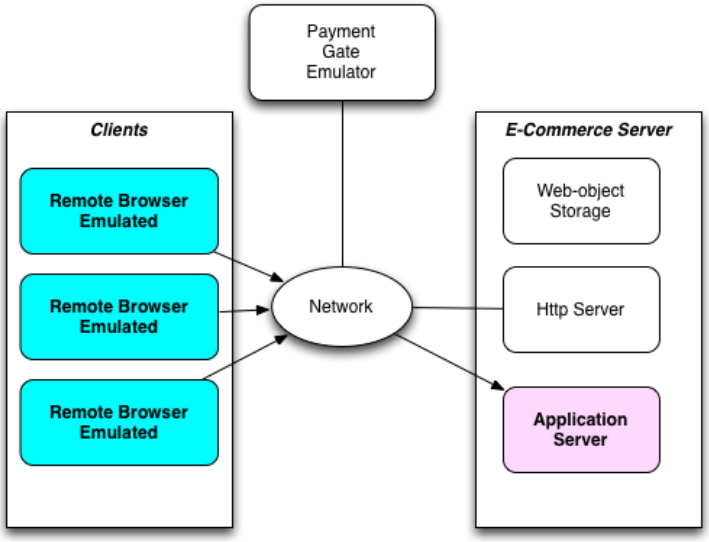


Figure A.2: TPC-W architecture (Mohammad S. Obaidat; ZARAI,) (MENASCÉ; MASON, 2002)

adding transactions and any other required logic (Fig. A.3). LoadRunner scripting only permits partial reproduction of user dynamism when generating Web workload, because it cannot define either advanced interactions of users, such as parallel browsing behavior, or continuous changes in user's behaviors (Mohammad S. Obaidat; ZARAI,).

WebLOAD is a software tool for Web performance commercialized by RadView. It is oriented to explore the performance of critical Web applications by quantifying the utilization of the main server resources. The tool creates scenarios that try to mimic the navigation of real users. To this end, it provides facilities to record, edit and debug test scripts, which are used to define the scenarios on workload characterization. The execution environment is a console to manage test execution, whose results are analyzed in the Analytics application. Since

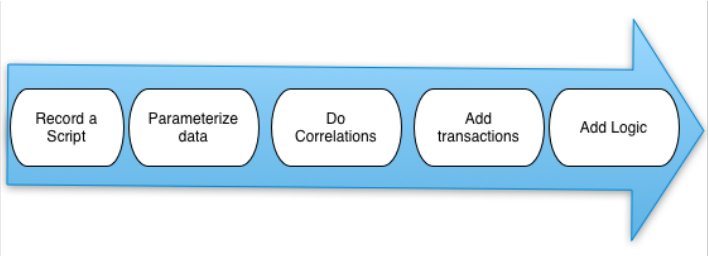


Figure A.3: Load Runner Scripting

WebLOAD is a distributed system, it is possible to deploy several load generators to reproduce the desired load. Load generators can also be used as probing clients where a single virtual user is simulated to evaluate specific statistics of a single user. These probing clients resemble the experience of a real user using the system while it is under load (Mohammad S. Obaidat; ZARAI,).

A.3.5 Cloud Testing Tools

Nachiyappan and Justus show a set of other tools perform Cloud testing. Cloud testing is a form of evaluation methodology in which the applications to be tested uses the cloud as a computing environment and its infrastructure to simulate real world traffic by using existing cloud computing technologies. Cloud testing is challenged by several problems such as limited budget, meeting deadlines, High cost per test, a large number of test cases, little reuse of tests and geographical distributions of users. Blitz is a load-testing tool from the cloud to the cloud. Blitz has no client to install and it is unable to test applications behind firewalls or otherwise protected from the Internet. Blaze Meter is a cloud application based on JMeter scripts that allow stress and load tests on the cloud (NACHIYAPPAN; JUSTUS, 2015).

SOASTA CloudTest is a production performance testing tool for Web applications. It can simulate thousands of virtual users visiting website simultaneously, using either private or public cloud infrastructure service. The worker nodes can be distributed across public and private clouds to cooperate in a large load testing. Test results from distributed test agents are integrated for analysis (BAI et al., 2011).

A.3.6 Apache JMeter

Apache JMeter was the tool chosen for current research due to its open license, the use of plugins and the ease of integration with jmetal and jgap frameworks. Apache JMeter is a free open source stress testing tool. It has a large user base and offers lots of plugins to aid testing.

JMeter is a desktop application designed to test and measure the performance and functional behavior of applications. The application it is purely Java-based and is highly extensible through a provided API (Application Programming Interface). JMeter works by acting as the client of a client/server application. JMeter allows multiple concurrent users to be simulated on the application (HALILI, 2008) (ERINLE, 2013).

Apache JMeter is used to generate heavy loads on the servers or objects to test its strength or analyze overall performance under different load types. To briefly explain the solution how it works, as follows: A Regular Expression Extractor captures the dynamic values as mentioned above and stored in a temporary variable. The values which have been extracted and stored in temporary variables are subsequently utilized by immediate requests/re directions using HTTP samplers (KIRAN; MOHAPATRA; SWAMY, 2015). JMeter has components organized in a hierarchical manner. The Test Plan is the main component in a JMeter script. A typical test plan will consist of one or more Thread Groups, logic controllers, listeners, timers, assertions, and configuration elements:

- Thread Group: Test management module responsible for emulating the users used in a test. All elements of a test plan must be under a thread group.
- Listeners: Analysis module responsible for providing access to the information gathered by JMeter about the test cases.
- Samplers: Load injectors module responsible for sending requests to a server, while Logical Controllers let you customize its logic.
- Timers: allow JMeter to delay between each request.
- Assertions: test if the application under test it is returning the correct results.
- Configuration Elements: configure details about the request protocol and test elements.

A.4 Research Question 1: How to model a stress test workload?

The design of a stress test depends intrinsically on the load model applied to the software under test. Based on the objectives, there are two general schools of thought for designing a proper load to achieve such objectives (AFZAL; TORKAR; FELDT, 2009):

- Designing Realistic Loads (Descriptive Workload).

•Designing Fault-Inducing Loads (Generative Workload).

In Designing Realistic Loads, the main goal of testing is to ensure that the system can function correctly once. Designing Fault-Inducing Loads aims to design loads, which are likely to cause functional or non-functional problems (AFZAL; TORKAR; FELDT, 2009).

Stress testing projects should start with the development of a model for user workload that an application receives. This should take into consideration various performance aspects of the application and the infrastructure that a given workload will impact. A workload is a key component of such a model (MOLYNEAUX, 2009).

The term workload represents the size of the demand that will be imposed on the application under test in an execution. The metric used to measure a workload is dependent on the application domain, such as the length of the video in a transcoding application for multimedia files or the size of the input files in a file compression application (FEITELSON, 2013) (MOLYNEAUX, 2009) (GONçALVES, 2014).

The workload is also defined by the load distribution between the identified transactions at a given time. Workload helps researchers study the system behavior identified in several load models. A workload model can be designed to verify the predictability, repeatability, and scalability of a system (FEITELSON, 2013) (MOLYNEAUX, 2009). Workload modeling is the attempt to create a simple and generic model that can then be used to generate synthetic workloads. The goal is to create workloads that can be used in performance evaluation studies. Sometimes, the synthetic workload is supposed to be similar to those that occur in practice in real systems (FEITELSON, 2013) (MOLYNEAUX, 2009).

There are two kinds of workload models: descriptive and generative. The main difference between the two is that descriptive models just try to mimic the phenomena observed in the workload, whereas generative models try to emulate the process that generated the workload in the first place (FEITELSON, 2013).

In descriptive models, one finds different levels of abstraction on the one hand and different levels of fidelity to the original data on the other hand. The most strictly faithful models try to mimic the data directly using the statistical distribution of the data. The most common strategy used in descriptive modeling is to create a statistical model of an observed workload. This model is applied to all the workload attributes, e.g., computation, memory usage, I/O behavior, communication, etc. (FEITELSON, 2013). Fig. A.4 shows a simplified workflow of a descriptive model. The workflow has six phases. In the first phase, the user uses the system in the production environment. In the second phase, the tester collects the user's data, such as

logs, clicks, and preferences, from the system. The third phase consists of developing a model designed to emulate the user's behavior. The fourth phase is made up of the execution of the test, emulation of the user's behavior, and log gathering.

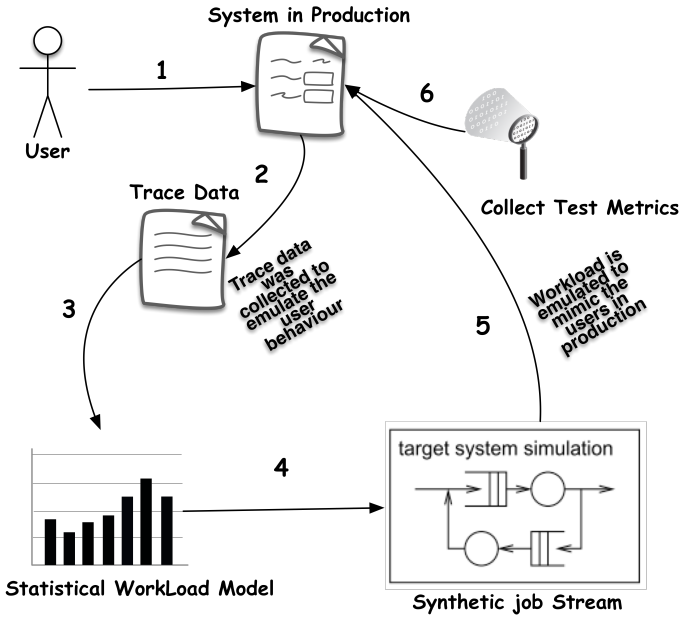


Figure A.4: Workload modeling based on statistical data (Di Lucca; FASOLINO, 2006)

Generative models are indirect in the sense that they do not model the statistical distributions. Instead, they describe how users will behave when they generate the workload. Consequently, an important benefit of the generative approach is that it facilitates manipulations of the workload. It is often desirable to be able to change the workload conditions as part of the evaluation. Descriptive models do not offer any option regarding how to do so. With the generative models, however, we can modify the workload-generation process to fit the desired conditions (FEITELSON, 2013). The difference between the workflows of the descriptive and the generative models is that user behavior is not collected from logs but simulated from a model that can receive feedback from the test execution (Fig. A.5).

Both load models have their advantages and disadvantages. In general, loads resulting from realistic-load based design techniques (Descriptive models) can be used to detect both functional and non-functional problems. However, the test duration is usually longer and the test analysis is more difficult. Loads resulting from fault-inducing load design techniques (Generative models) take less time to uncover potential functional and non-functional problems, where the resulting loads usually only cover a small portion of the testing objectives (JIANG, 2010). The presented research work uses a generative model.

There are two main approaches to design generative or descriptive workloads:

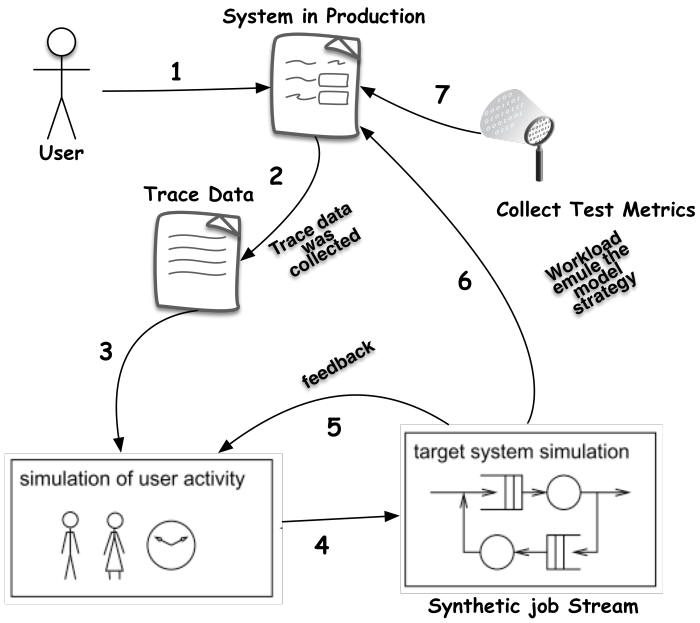


Figure A.5: Workload modeling based on the generative model (Di Lucca; FASOLINO, 2006)

•Model-based Stress testing: a usage model is proposed to simulate users' behaviors.

•Search-based Stress testing.

A model-based test is an approach that automates test case generation and execution in order to reduce software testing cost. Search-based techniques are commonly used to generate optimal test data that satisfies every transition and constraint in the state model (SAEED; Ab Hamid; SANI, 2017).

Search-Based Stress testing will be detailed explained in chapter 3. Three other approaches use neither a model-based test nor a search-based test.

A.4.1 Model-based Stress Testing

Model-based testing is an application of models to represent the desired behavior of a System Under Test or to represent testing strategies in a test. Some research approaches propose models to simulate or generate realistic loads. Model-based testing (MBT) is a variant of testing that relies on explicit behavior models that encode the intended behaviors of a system under test. Test cases are generated from one of these models or their combination (Mark Utting; LEGEARD, 2012) (MODEL-BASED..., 2008). There are many different modeling notations that have been used for modeling the behavior of systems for test generation purposes (Mark Utting; LEGEARD, 2012) (HIERONS et al., 2009).

•State-Based (or Pre/Post) Notations. These model a system as a collection of variables, which represent a snapshot of the internal state of the system, plus some operations that modify those variables. Each operation is usually defined by a pre-condition and a post-condition, or the post-condition may be written as explicit code that updates the state (Mark Utting; LEGEARD, 2012).

•Transition-based Notations. These focus on describing the transitions between different states of the system. Typically, they are graphical node-and-arc notations, like finite state machines (FSMs). Examples of transition-based notations used for MBT include FSMs themselves, state-charts, labeled transition systems and I/O automata (Mark Utting; LEGEARD, 2012).

•History-based Notations. These notations model a system by describing the allowable traces of its behavior over time. Message-sequence charts and related formalism are also included in this group. These are graphical and textual notations for specifying sequences of interactions between components (Mark Utting; LEGEARD, 2012).

•Functional Notations. These describe a system as a collection of mathematical functions. The functions may be first-order only, as in the case of algebraic specifications, or higher-order, as in notations like HOL (Mark Utting; LEGEARD, 2012). Functional models also show the functionality of the system from the user's perspective (YE, 2007). This research also classified in the functional notation paradigm the studies that used more than one UML diagram.

•Operational Notations. These describe a system as a collection of executable processes, executing in parallel. They are particularly suited to describing distributed systems and communications protocols. Examples include process algebras such as CSP or CCS as well as Petri net notations. Slightly stretching this category, hardware description languages like VHDL or Verilog are also included in this category (Mark Utting; LEGEARD, 2012).

•Stochastic Notations. These describe a system by a probabilistic model of the events and input values and tend to be used to model environments rather than SUTs. For example, Markov chains are used to model expected usage profiles, so that the generated tests scenarios (Mark Utting; LEGEARD, 2012).

•Data-Flow Notations. These notations concentrate on the data rather than the control flow. Prominent examples are Lustre, and the block diagrams of Matlab Simulink, which are often used to model continuous systems (Mark Utting; LEGEARD, 2012).

Table A.3 presents the papers found by the survey about model-based tests. All results were classified by model and paradigm. The most used paradigms in model-based stress testing are Functional-based notations, Stochastic notations, Transition-based and State-based notations.

Functional Notation

All possible answers of the system, including exceptions, are defined in the functional model. The functional notation defines the authorized input values and models all the possible functional errors during execution (UTTING; LEGEARD, 2010). Among the several functional model's approaches, we can highlight the User Community Modeling Language (UCML). A UCML is a set of symbols that can be used to create visual system usage models and depict associated parameters (WANG; ZHOU; LI, 2013). The Fig. A.6 shows a sample where all users realize a login into the application under test. Once logged in, 40% of the users navigate to the application, 30% of the users realize downloads. 20% of users realize uploads and 10% of users perform deletions.

Garousi et al. propose derive Stress Test Requirements from a UML model. The input model consists of a number of UML diagrams. Some of them are standard in mainstream development methodologies and others are needed to describe the distributed architecture of the system under test (Fig. A.9). Cai and Zeng use activity diagrams to describe variation points in use cases. Variation points describe what varies between the applications of a software product line (Xinying Cai, 2007). Raulf et al. present an approach for testing of web service compositions using UML profile for Business Process Execution Language (BPEL) (RAUF; IQBAL; MALIK, 2009). Schaefer et al. present the Crushinator, a framework that provides a game-independent testing tool simulating clients that perform HTTP requests using a UML model (SCHAEFER; DO; SLATOR, 2013). Moscher and Fögen compare the techniques Capture and Replay (CR) and Model-Based Testing (MBT) are using a model named PLeTsPerf. PLeTsPerf describes the system under tests using use cases and activity diagrams (MOSCHER; FÖGEN, 2017). Kim proposes an approach to generate massive virtual clients and realistic traffic. A session is composed of a series of the expected operations made by a user in the target domain using a UML class and a sequence diagram (KIM, 2005).

Stochastic Notation

For many software programs, probabilistic models are a useful asset in modeling statistical behavior, such that coverage testing is possible by automating test-case selection, execution and evaluation. Usually, the stochastic notations are used in Markov Chains and Stochastic

Formcharts.

Avritzer and Weyuker present two variants Markov chain approach to realize load and stress tests and an automatic generation of load test suites approach (AVRITZER; LARSON, 1993) (AVRITZER; WEYUKER, 1995). Barros et al. provide techniques for load pattern characterization via the application of Markov Chains to performance evaluation of stateful systems (BARROS; SHIAU, 2007). Cai et al. use a stochastic form chart as its client loading model (CAI; GRUNDY; HOSKING, 2007). The work of Draheim and Weber's Form-oriented analysis is a methodology for the specification of ultra-thin client based systems. Form-oriented models describe a web application as a bipartite state machine which consists of pages, actions, and transitions between them. Stochastic Formcharts are the combination of form-oriented model and probability features. The Fig. A.7 shows a sample where all users have a probability of 100% of realizing a login into the application under test. Once logged in, users have a probability of 40% of navigating on the application and so on (DRAHEIM et al., 2006) (MARINESCU et al., 2015).

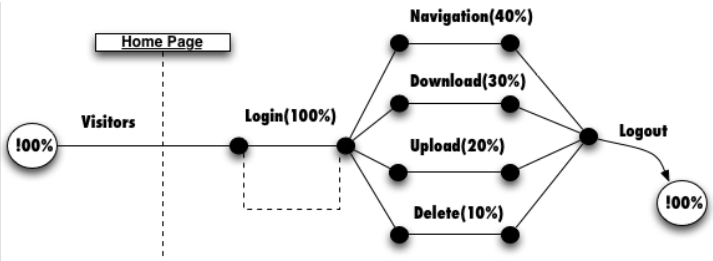


Figure A.6: User community modeling language (WANG; ZHOU; LI, 2013)

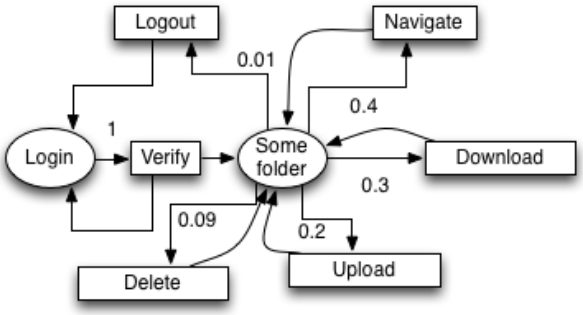


Figure A.7: Stochastic Formcharts Example (DRAHEIM et al., 2006) (WANG; ZHOU; LI, 2013)

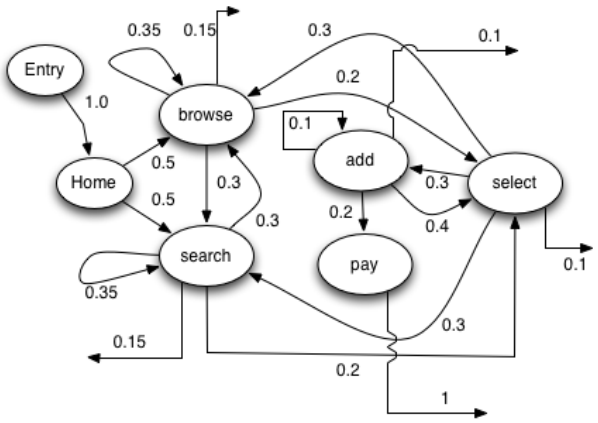


Figure A.8: Example of a Customer Behavior Model Graph (CBMG) (MENASCÉ; MASON, 2002) (JIANG, 2010) (Mohammad S. Obaidat; ZARAI,)

One way to capture the navigational pattern within a session is through the Customer Behavior Model Graph (CBMG). Figure A.8 depicts an example of a CBMG showing that customers

may be in several different states—Home, Browse, Search, Select, Add, and Pay—and they may transition between these states as indicated by the arcs connecting them. The numbers on the arcs represent transition probabilities. A state not explicitly represented in the figure is the Exit state (MENASCÉ; MASON, 2002) (JIANG, 2010) (Mohammad S. Obaidat; ZARAI,).

Transition-based notation

For model-based testing, the transition-based notations are the most used for developing behavioral models (UTTING; LEGEARD, 2010). Broadly speaking, the transition-based notations are best for control-oriented applications. Instead of characterizing the system based on its admissible states, the system is characterized as transitions from one state to another, the properties are specified as a set of transitions functions, which map each input state to the corresponding output state. Based on the notations used, the model can be annotated with triggering events, which are conditions sufficient for the transition to take place, or guards that are necessary preconditions for the transition to be fired. The common techniques used for generating test-cases from transition-based notations are Finite State Machines (FSM), Labeled Transition Systems and UML state-charts (MARINESCU et al., 2015).

Arantes et al. present a tool named WEB-PerformCharts that generate test cases using state-charts or FSMs (ARANTES et al., 2014). Gay et al. present propose an automated steering framework that can adjust the behavior of the model to better match the behavior of the system under test to reduce the rate of false positives. The model describes as a transition system (GAY; RAYADURGAM; HEIMDAHL, 2016). Hessel addressed in her study two model-based problems: how to formalize coverage criteria and how to generate a test suite to a formal timed system model (HESSEL, 2007). Ganesan describes how we created a test automation architecture for testing concurrent inter-task communication as carried out by the software bus. The model uses an FSM (GANESAN et al., 2016). Hierons et al. show a formal specification approach to support functional or non-functional tests (HIERONS et al., 2009). Vogele et al. present an approach that aims to automate the extraction and transformation of workload specifications for a model-based performance prediction of session-based application systems. The research also presents transformations to the common load testing tool Apache JMeter and to the Palladio Component Model (VOGELE et al., 2016) (UTTING; LEGEARD, 2010). The workload specification formalism (Workload Model) consists of the following components, which are detailed below and illustrated in Fig. A.10:

- An Application Model, specifying allowed sequences of service invocations and SUT-specific details for generating valid requests.

- A set of Behavior Models, each providing a probabilistic representation of user sessions in terms of invoked services.

- A Behavior Mix, specified as probabilities for the individual Behavior Models to occur during workload generation.

- A Workload Intensity that includes a function which specifies the number of concurrent users during the workload generation execution.

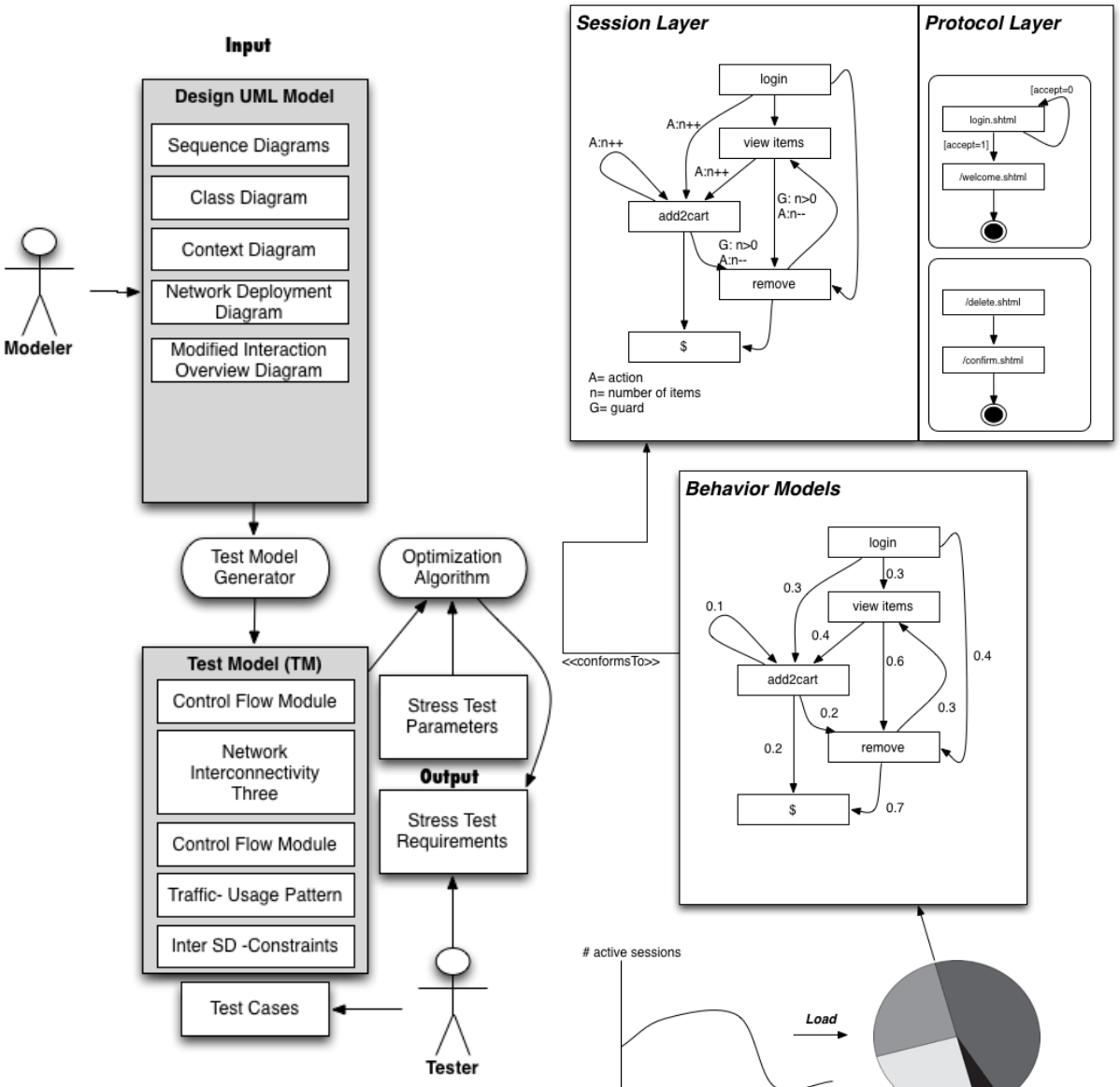


Figure A.9: Model-based stress test methodology

Figure A.10: Exemplary workload model

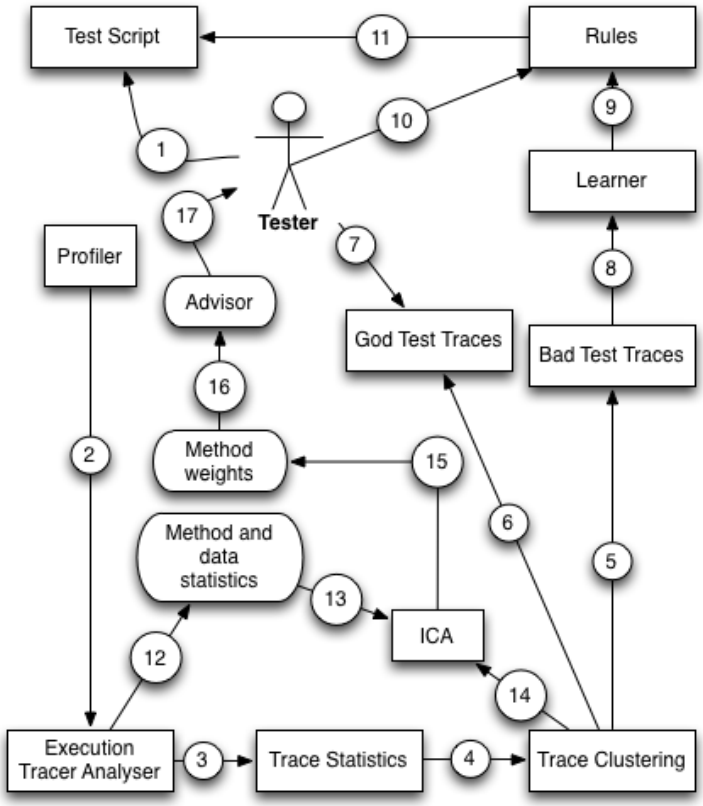


Figure A.11: The architecture and workflow of FOREPOST

State-based notation

State based notations like STATECHARTS or activity diagrams describe the behavior based on abstract states and support behavior integration only using state composition (GIESE; GRAF; WIRTZ, 1999). In State-based or notations, the system is modeled as a collection of variables representing its state at a specific point of the execution, together with a collection of operations defined by a precondition that defines the admissible set of initial states, and a postcondition that specifies the guaranteed set of final states. Examples of such notations include the Z language, the B machine, UML's Object Constraint Language (OCL), Java Modeling Language (JML), VDM , and Spec# (MARINESCU et al., 2015).

Sridhar proposed an approach to generate test cases with MATLAB using Simulink/ Stateflow tool. After the model creation, test sequences are generated a dependency graph of that system (SRIDHAR; SRINIVASULU; MOHAPATRA, 2013). Fang et al. developed a test case generator, from which an entire test suite can be extracted (FANG et al., 2012). Jeong et al. propose a state transition model based to test case generation (JEONG et al., 2016). Wieczorek et al. propose an approach that uses proprietary models called Message Choreography Models (MCM) using a state-based representation (WIECZOREK; STEFANESCU; ROTH, 2010).

A.4.2 Other Approaches

A set of other approaches don't use model-based tests or search-based tests:

- Automatic feedback, control-based, stress and load testing (BAYAN; CANGUSSU, 2008);
- Feedback-ORiEnted PerfOrmance Software Testing (LUO et al., 2015);
- PASASM: A Method for the Performance Assessment of Software Architectures (WILLIAMS; SMITH, 2002).

Bayan and Cangussu present an approach based on the application of a feedback PID (Proportional, Integral, and Derivative) controller to drive the input and make the system achieve a specified level of resource usage. For example, if the user defines the system should be tested with a memory use of 95%, starting from an initial input value, the PID controller will automatically change the input(s) until the desired level of stress has been achieved (BAYAN; CANGUSSU, 2008).

Williams and Smith describe the PASA method, a method for performance assessment of software architectures. PASA uses the principles and techniques of software performance engineering (SPE) to determine whether an architecture is capable of supporting its performance objectives (WILLIAMS; SMITH, 2002). Among the three approaches presented, we can highlight FOREPOST that develops a plugin of the JMeter tool to generate test cases using unsupervised learning.

Feedback-ORiEnted PerfOrmance Software Testing

Feedback-ORiEnted PerfOrmance Software Testing (FOREPOST) is an adaptive, feedback-directed learning testing system that learns rules from system execution traces and uses these learned rules to select test input data automatically to find more performance problems in applications when compared to exploratory random performance testing (GRECHANIK; FU; XIE, 2012).

FOREPOST uses runtime monitoring for a short duration of testing together with machine learning techniques and automated test scripts to reduce large amounts of performance-related information collected during AUT runs to a small number of descriptive rules that provide insights into properties of test input data that lead to increased computational loads of applications.

The Fig. A.11 presents the main workflow of FOREPOST solution. The first step, The Test Script is written by the test engineer(1). Once the test script starts, its execution traces are collected (2) by the Profiler, and these traces are forwarded to the Execution Trace Analyzer, which produces (3) the Trace Statistics. The trace statistics is supplied (4) to Trace Clustering, which uses an ML algorithm, JRip to perform unsupervised clustering of these traces into two groups that correspond to (6) Good and (5) Bad test traces.

The user can review the results of clustering (7). These clustered traces are supplied (8) to the Learner that uses them to learn the classification model and (9) output rules. The user can review (10) these rules and mark some of them as erroneous if the user has sufficient evidence to do so. Then the rules are supplied (11) to the Test Script. Finally, the input space is partitioned into clusters that lead to good and bad test cases, to find methods that are specific to good performance test cases. This task is accomplished in parallel to computing rules, and it starts when the Trace Analyzer produces (12) the method and data statistics that are used to construct (13) two matrices (14). Once these matrices are constructed, ICA[1] decomposes them (15) into the matrices for bad and good test cases correspondingly. Finally, the Advisor (16) determines top methods that performance testers should look at (17) to debug possible performance problems.

A.5 Research Question 2: What are the main anti-patterns found by stress tests?

Performance problems share common symptoms and many performance problems described in the literature are defined by a particular set of root causes. Fig. A.12 shows the symptoms of known performance problems (WERT; HAPPE; HAPPE, 2013).

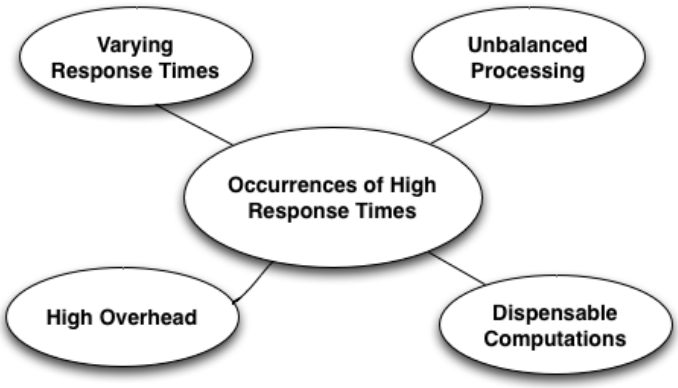


Figure A.12: Symptoms of known performance problems (WERT; HAPPE; HAPPE, 2013).

[1]ICA is a technique to separate linearly mixed sources (LUO et al., 2015)

There are several anti-patterns that detailed features about common performance problems. Antipatterns are conceptually similar to patterns in that they document recurring solutions to common design problems. They are known as antipatterns because their use produces negative consequences.

Performance anti-patterns document common performance mistakes made in software architectures or designs. These software Performance anti-patterns have four primary uses: identifying problems, focusing on the right level of abstraction, effectively communicating their causes to others and prescribing solutions (BROWN et al., 1998). Table A.4 present some of the most common performance anti-patterns.

Blob anti-pattern is known by various names, including the “god” class [8] and the “blob” [2]. Blob is an anti-pattern whose problem is on the excessive message traffic generated by a single class or component, a particular resource does the majority of the work in a software. The Blob occurs when a single class or component either performs all of the work of an application or holds all of the application’s data. Either manifestation results in excessive message traffic that can degrade performance (CORTELLESSA; FRITTELLA, 2007) (SMITH; WILLIAMS, 2000).

A project containing a “god” class usually has a single, complex controller class that is surrounded by simple classes that serve only as data containers. These classes typically contain only accessor operations (operations to get() and set() the data) and perform little or no computation of their own (SMITH; WILLIAMS, 2000). According to Figure A.13 and A.14, a hypothetical system with a BLOB problem is shown: Figure A.13 presents a sample where the Blob class uses the features A,B,C,D,E,F and G of the hypothetical system, and Fig. A.14 shows a static view where a complex software entity instance, i.e. Sd, is connected to other software instances, e.g. Sa, Sb and Sc, through many dependencies (TRUBIANI, 2011b)(WERT; HAPPE; HAPPE, 2013).

A characteristic of Unbalanced Processing is the tendency to overload a particular resource, wherein one scenario a specific class of requests generates a pattern of execution within the system. In other words, the overloaded resource will be executing a certain type of job very often, thus in practice damaging other classes of jobs that will experience very long waiting times. Unbalanced Processing occurs in three different situations. The first case that causes unbalanced processing is when processes cannot make effective use of available processors either because processors are dedicated to other tasks or because of single-threaded code. This manifestation has available processors and we need to ensure that the software is able to use them. Fig. A.15 shows a sample of the Unbalanced Processing. In Fig. A.15, four tasks

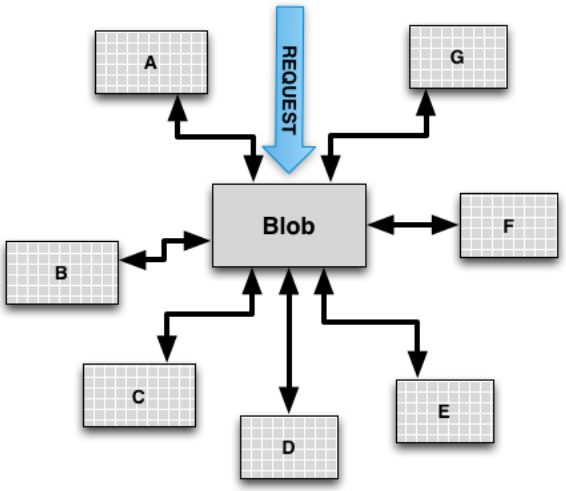


Figure A.13: The God class(WERT; HAPPE; HAPPE, 2013).

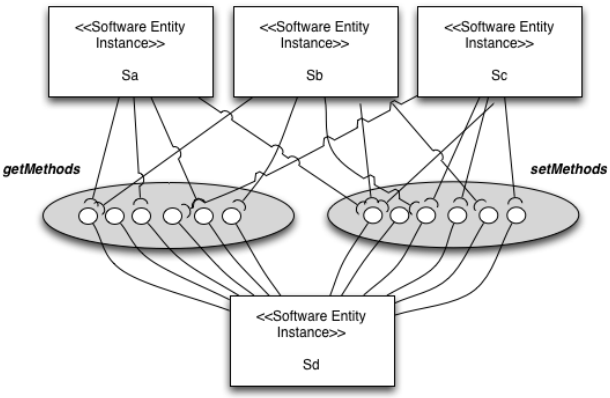


Figure A.14: The God class(TRUBIANI, 2011b).

are performed. Task D is waiting for task C to conclude, and both are submitted to a heavy processing situation.

The pipe and filter architectures and extensive processing anti-pattern represent a manifestation of the unbalanced processing anti-pattern. The pipe and filter architectures occur when the throughput of the overall system is determined by the slowest filter. Fig A.16 illustrates a software S with a Pipe and Filter Architectures problem: the operation opx is invoked in a service and the throughput of the service ($Th(S)) is lower than the required one. The extensive processing occurs when a process monopolizes a processor and prevents a set of other jobs to be executed until it finishes its computation. The Fig. A.17 describes a software S with an Extensive Processing problem: the operations opx and opy are alternatively invoked in a service and the response time of the service ($RT(S)) is larger than the required one (TRUBIANI, 2011b).

Circuitous Treasure Hunt anti-pattern occurs when software retrieves data from a first component, uses those results in a second component, retrieves data from the second component, and so on until the last results are obtained (SMITH; WILLIAMS, 2002) (SMITH; WILLIAMS, 2003). Circuitous Treasure Hunt are typical performance anti-patterns that causes unnecessarily high amount of frequent database requests. The Circuitous Treasure Hunt anti-pattern is a result of a bad database schema or query design. A common Circuitous Treasure Hunt design creates a data dependency between single queries. For instance, a query requires the result of a previous query as input. The longer the chain of dependencies between individual queries the more the Circuitous Treasure Hunt hurts performance (WERT et al., 2014). Fig. A.18 shows a software S with a Circuitous Treasure Hunt problem: the software S generates a large number of database calls by performing several queries up to the final operation (TRUBIANI, 2011b).

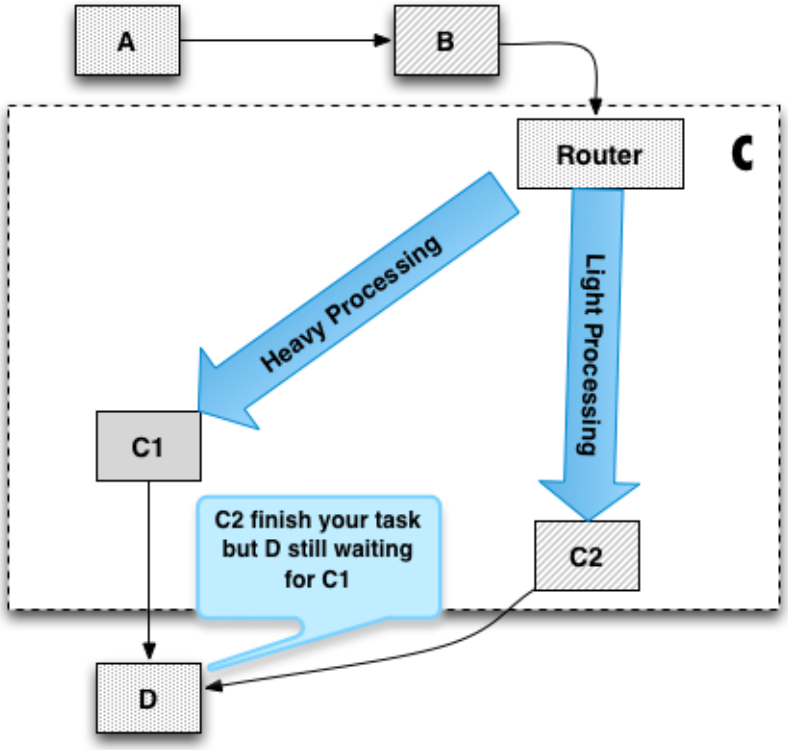


Figure A.15: Unbalanced Processing sample (WERT; HAPPE; HAPPE, 2013).

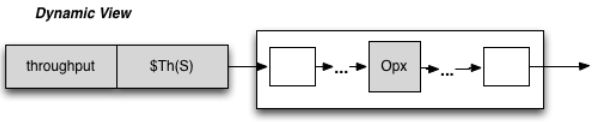


Figure A.16: Pipe and Filter sample (TRUBIANI, 2011b)

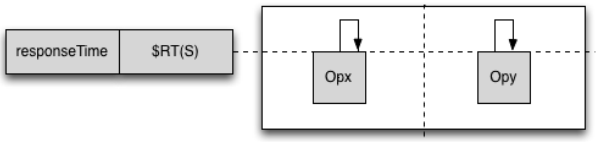


Figure A.17: Extensive Processing sample (TRUBIANI, 2011b).

Empty Semi Trucks occurs when an excessive number of requests is required to perform a task. It may be due to inefficient use of available bandwidth, an inefficient interface, or both (ARCELLI; CORTELLESSA; TRUBIANI, 2012). There is a special case of Empty Semi Trucks that occurs when many fields in a user interface must be retrieved from a remote system. Fig. A.19 shows a software S with an Empty Semi Trucks problem: the software instance Sa generates an excessive message traffic by sending a high amount of messages with low sizes, much lower than the network bandwidth, hence the network link might have a low utilization value (TRUBIANI, 2011b).

The Tower of Babel anti-pattern most often occurs when information is translated into an exchange format, such as XML, where the sending process is then parsed and translated into an internal format by the receiving process. When the translation and parsing are excessive, the system spends most of its time doing this and relatively little real work (SMITH; WILLIAMS,

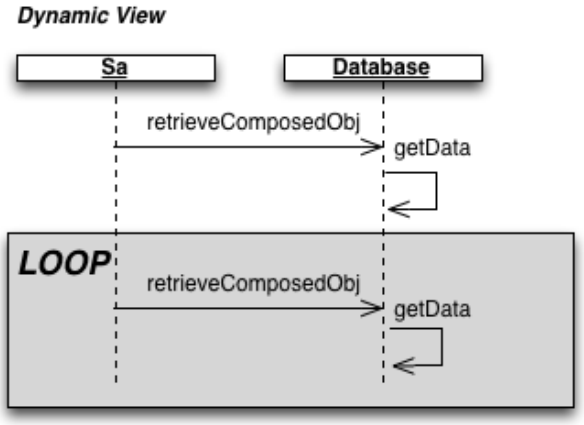


Figure A.18: Circuitous Treasure Hunt sample (TRUBIANI, 2011b)

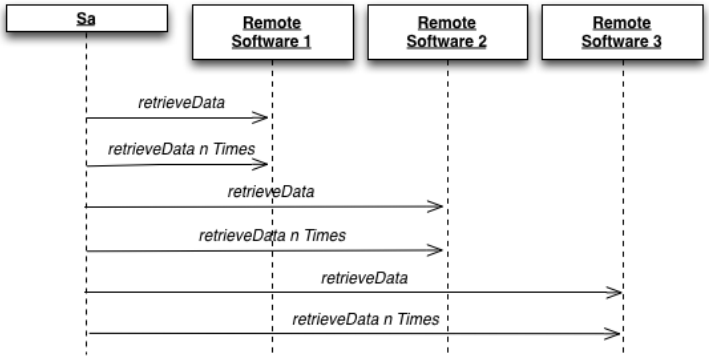


Figure A.19: Empty Semi Trucks sample (TRUBIANI, 2011b).

2003). Fig. A.20 shows a system with a Tower of Babel problem: the software instances Sd performs the format translation to communicate with other instances many times (TRUBIANI, 2011b).

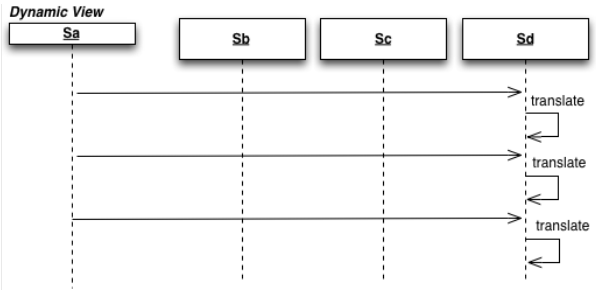


Figure A.20: Tower of Babel sample (TRUBIANI, 2011b)

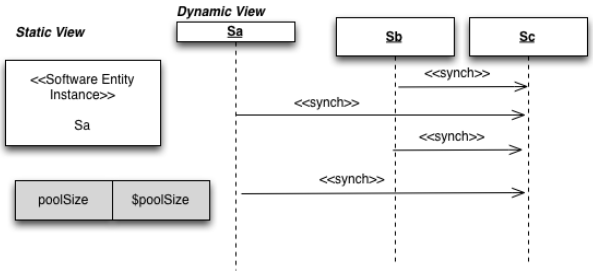


Figure A.21: One-Lane Bridge sample (TRUBIANI, 2011b).

One-Lane Bridge is an anti-pattern that occurs when one or a few processes execute concurrently using a shared resource and while the other processes are waiting to use the shared resource. It frequently occurs in applications that access a database. Here, a lock ensures that only one process may update the associated portion of the database at a time. This anti-pattern is common when many concurrent threads or processes are waiting for the same shared resources. These can either be passive resources (like semaphores or mutexes) or active resources (like CPU or hard disk). In the first case, we have a typical One Lane Bridge whose critical resource needs to be identified. Figure 3.10 shows a system with a One-Lane Bridge problem: the software instance Sc receives an excessive number of synchronous calls in a service S and the predicted response time is higher than the required one (TRUBIANI, 2011b).

Using dynamic allocation, objects are created when they are first accessed and then destroyed when they are no longer needed. Excessive Dynamic Allocation, however, addresses

frequent, unnecessary creation and destruction of objects of the same class. Dynamic allocation is expensive, an object created in memory must be allocated from the heap, and any initialization code for the object and the contained objects must be executed. When the object is no longer needed, necessary cleanup must be performed, and the reclaimed memory must be returned to the heap to avoid memory leaks (SMITH; WILLIAMS, 2002) (SMITH; WILLIAMS, 2003).

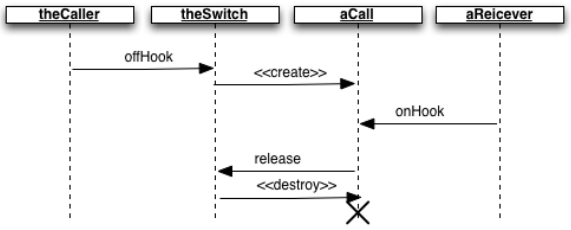


Figure A.22: Excessive Dynamic Allocation.

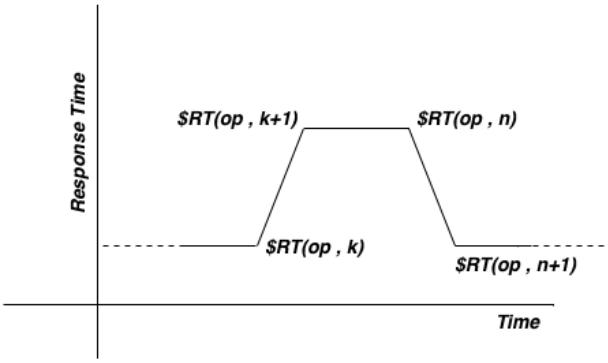


Figure A.23: Traffic Jam Response Time (TRUBIANI, 2011b).

Fig. A.22 shows an Excessive Dynamic Allocation sample. This example is drawn from a call (an offHook event) and the switch creates a Call object to manage the call. When the call is completed, the Call object is destroyed. Constructing a single Call object is not as excessive. A Call is a complex object that contains several other objects that must also be created. The Excessive Dynamic Allocation occurs when a switch receives hundreds of thousands of offHook events. In a case like this, the overhead for dynamically allocating call objects adds substantial delays to the time needed to complete a call.

The Traffic Jam anti-pattern occurs if many concurrent threads or processes are waiting for the same active resources (like CPU or hard disk). This anti-pattern produces a large backlog in jobs waiting for service. The performance impact of the Traffic Jam is the transient behavior that produces wide variability in response time. Sometimes it is fine, but at other times, it is unacceptably long. Figure A.23 exhibits a software with a Traffic Jam problem, in which the monitored response time of the operation shows a wide variability in a response time that persists long (TRUBIANI, 2011b).

Ramp is an anti-pattern where the processing time increases as the system is used. 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 anti-pattern, the memory consumption of the application is growing over time. The root causes are the Specific Data Structures, which are growing during operation or which are not properly disposed (WERT

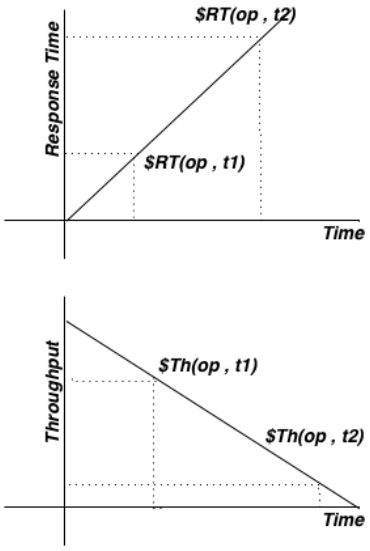


Figure A.24: The Ramp sample (TRUBIANI, 2011b).

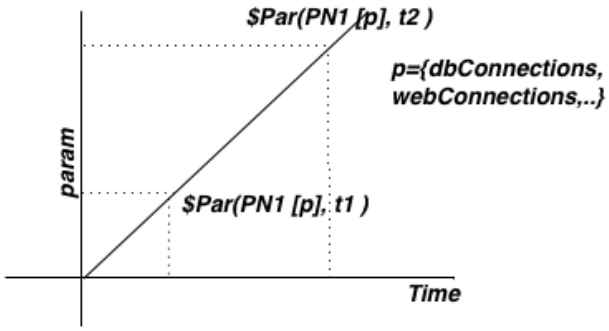


Figure A.25: More is Less sample (TRUBIANI, 2011b).

et al., 2014) (SMITH; WILLIAMS, 2003). Fig. A.24 shows a system with the Ramp problem: (i) the monitored response time of the operation opx at time t1, i.e. $RT(opx, t1), is much lower than at time t2, i.e. $RT(opx, t2), with t1 < t2; (ii) the monitored throughput of the operation opx at time t1, i.e. $Th(opx, t1), is much larger than at time t2, i.e. $Th(opx, t2), with t1 < t2.

More is Less occurs when a system spends more time "thrashing" than accomplishing real work because there are too many processes relative to available resources. More is Less is presented when it is running too many programs over time. This anti-pattern causes too much system paging and systems spend all their time servicing page faults rather than processing requests. In distributed systems, there are more causes. They include: creating too many database connections and allowing too many internet connections. Fig. A.25 illustrates a system with a More Is Less problem: There is a processing node PN1 and the monitored runtime parameters (e.g. database connections, etc.) at time t1, i.e. $Par(PN1[p], t1), are much larger than the same parameters at time t2, i.e. $Par(PN1[p], t2), with t1 < t2.

A.6 Summary

Stress testing investigates the behavior of the system under conditions that overload its resources (SANDLER; BADGETT; THOMAS, 2004) (LEWIS; DOBBS; VEERAPILLAI, 2005). The core activities in conducting a usual Load, Performance and Stress tests are Identify the test environment, Identify acceptance criteria, Plan and design tests, Prepare the test environment, Record the test plan, Run the tests and Analyze results. This research focuses on the generation

and execution of test cases. It is not the scope of the research the automatic analysis of the results and the treatment of log of tests. The two main approaches to creating workloads are generative and descriptive workload approaches. The main goal of descriptive workloads is to ensure that the system can function correctly once. Generative workloads aims to design loads, which are likely to cause a functional or non-functional problem.

There are three approaches to stress test execution: Live-User Based Executions, Driver Based Executions and Emulation Based Executions. This thesis focuses on user Driver Based Executions where each workload performs several requests to an application under test to measure the response time. Apache JMeter was the tool chosen for current research due to its open license, the use of plugins and the ease of integration with jmetal and jgap frameworks.

The main approaches to generate test cases to stress testing are: Model-based testing and Search-based testing. Model-based testing uses a model to describe the states and transitions of a system under test. Although the use of models allows greater control of generated test cases, since all test cases are generated from the previously defined model, it is also the main limitation of this approach since the model may not include all possible scenarios of the test because it depends intrinsically on the test designer responsible for creating the models. The most used paradigms in model-based stress testing are: Functional-based notations, Stochastic notations, Transition-based and State-based notations.

There are several anti-patterns that detailed features about common performance problems. Antipatterns are conceptually similar to patterns in that they document recurring solutions to common design problems. This survey found 10 anti-patterns of which 4 were implemented by IAdapter in an experiment presented in chapter 4.

Table A.3: Summary of studies in model-based stress testing

Model	Paper	Paradigm	Year
BeliefDesire-Intention	(ARAIZA-ILLAN; PIPE; EDER, 2016)	Operational	2016
Markov-Chains	(AVRITZER; WEYUKER, 1995)	Stochastic Notation	1995
	(BARROS; SHIAU, 2007)	Stochastic Notation	2007
	(AVRITZER; WEYUKER, 1994)	Stochastic Notation	1994
	(AVRITZER; LARSON, 1993)	Stochastic Notation	1993
MCM Model	(WIECZOREK; STEFANESCU; ROTH, 2010)	State-based	2010
Other Approaches	(ENOIU; SUNDMARK; PETTERSSON, 2013)	UPPAAL model checker	2013
	(ARCAINI; GARGANTINI; RICCOBENE, 2015)	Model Decomposition	2015
Petri Nets	(BUCHS; LUCIO; CHEN, 2009)	Operational	2009
Stochastic Form Model	(CAI; GRUNDY; HOSKING, 2007)	Stochastic Notation	2007
	(MENASCÉ; MASON, 2002)	Stochastic Notation	2002
	(DRAHEIM et al., 2006)	Stochastic Notation	2006
State-Machine Models	(FANG et al., 2012)	State-based	2012
	(SRIDHAR; SRINIVASULU; MOHAPATRA, 2013)	State-based	2013
	(GANESAN et al., 2016)	Transition-based	2016
	(San Miguel; TAKADA, 2016)	-	2016
	(ARANTES et al., 2014)	Transition-based	2014
	(GAY; RAYADURGAM; HEIMDAHL, 2016)	Transition-based	2016
	(HESSEL, 2007)	Transition-based	2007
	(HIERONS et al., 2009)	Transition-based	2009
	(JEONG et al., 2016)	State-based	2016
An orchestrated survey of methodologies for automated software test case generation	(ANAND et al., 2013)	—	2013
Survey on Model-based Testing of Web Applications	(WANG; ZHOU; LI, 2013)	—	2013
UCML	(BARBER, 1999)	Functional	1999
UML	(Xinying Cai, 2007)	Functional	2007
	(RAUF; IQBAL; MALIK, 2009)	Functional	2009
	(SCHAEFER; DO; SLATOR, 2013)	Functional	2013
	(Matthias Beyer, Winfried Dulz, 2014)	Functional	2014
	(MOSCHER; FÖGEN, 2017)	Functional	2017
	(KIM, 2005)	Functional	2005
	(RODRIGUES et al., 2014)	Functional	2014
	(SILVEIRA; RODRIGUES; ZORZO, 2011)	Functional	2011
	(VOGELE et al., 2016)	Transition-based	2016
	(ALESIO; SEN, 2017)	Functional	2017
	(LENZ; CHIMIAK-OPOKA; BREU, 2007)	Transition-based	2007

Table A.4: Performance anti-patterns

Antipattern		Papers
Blob or The God Class		(WERT et al., 2014) (SMITH; WILLIAMS, 2000) (TRUBIANI, 2011a) (TRUBIANI, 2011b) (CORTELLESSA; FRITTELLA, 2007) (SMITH; WILLIAMS, 2003)
Circuitous Treasure Hunt		(WERT et al., 2014) (TRUBIANI, 2011a) (TRUBIANI, 2011b) (SMITH; WILLIAMS, 2003) (SMITH; WILLIAMS, 2002)
Empty Semi Trucks		(WERT et al., 2014) (TRUBIANI, 2011a) (ARCELLI; CORTELLESSA; TRUBIANI, 2012) (TRUBIANI, 2011b)
Excessive Dynamic Allocation		(TRUBIANI, 2011a) (TRUBIANI, 2011b) (SMITH; WILLIAMS, 2003) (SMITH; WILLIAMS, 2002)
More is Less		(TRUBIANI, 2011b) (TRUBIANI, 2011a) (SMITH; WILLIAMS, 2003)
One-Lane Bridge		(TRUBIANI, 2011b) (TRUBIANI, 2011a) (SMITH; WILLIAMS, 2003) (SMITH; WILLIAMS, 2002)
Stifle		(WERT et al., 2014)
The Ramp		(TRUBIANI, 2011a) (TRUBIANI, 2011b) (SMITH; WILLIAMS, 2003)
Tower of Babel		(TRUBIANI, 2011a) (TRUBIANI, 2011b)
Traffic Jam		(TRUBIANI, 2011b) (SMITH; WILLIAMS, 2003) (SMITH; WILLIAMS, 2002)
Unbalanced Processing		(CORTELLESSA; FRITTELLA, 2007) (TRUBIANI, 2011a) (SMITH; WILLIAMS, 2003)
	Concurrent Processing System	(TRUBIANI, 2011b) (SMITH; WILLIAMS, 2003)
	Piper and Filter	(TRUBIANI, 2011b) (SMITH; WILLIAMS, 2003)
	Extensive Process	(TRUBIANI, 2011b) (SMITH; WILLIAMS, 2003)
Unnecessary Processing		(SMITH; WILLIAMS, 2003)

APPENDIX B – ADAPTED SEDR

Software is pervasive, which raises the value of testing it (SANDLER; BADGETT; THOMAS, 2004). Various actions outside the application under test can cause high response times such as pagination, network usage or even a software upgrade. It is necessary a noisy reduction strategy in stress test in this situations. Noisy optimization is currently receiving increasing popularity for its widespread applications in engineering optimization problems, where the objective functions are often found to be contaminated with noisy (RAKSHIT; KONAR; DAS, 2017).

Standard Error Dynamic Resampling (SEDR), strategy has been employed for solving both noisy single and multi-objective evolutionary optimization problems. The working principle of SEDR is to add samples to a solution sequentially until the standard error of the objectives falls below a chosen threshold value (SIEGMUND; NG; DEB, 2013). It was proposed in (Di Pietro; WHILE; BARONE, 2004) for single-objective optimization problems. In this study we apply SEDR on multi-objective problems by aggregating all objective values to a scalar value. As aggregation the median of the objective standard errors is used. The strategy is concerned with the optimal allocation of sampling budget to a trial solution based on the noise strength at its corresponding position in the search space. The contamination level of noise is captured by the standard error of the mean fitness estimate of a trial solution. The SEDR algorithm is described in Alg. 10.

Algorithm 10 SEDR algorithm (SIEGMUND; NG; DEB, 2013)

1: **input :** Solution s
2: Draw $b_{min} \geq 2$ initial samples of s, F(s)
3: Calculate mean of the available fitness for each of the m objectives: $\mu_i(s)$, i=1,...,m
4: Calculate standard deviation: $\sigma_i = \sqrt{\frac{1}{n-1}\sum_n^{j=1}(F_j^i - \mu_i(s))^2}$
5: Calculate the standard error: $se_i(s) = \frac{\sigma_i}{\sqrt{n}}$
6: Calculate an aggregation of the standard errors $\overline{se}(s)$
7: Stop if $\overline{se}(s)$ > threshold or $b_s \geq b_{max}$ otherwise go to step 2

Numerous tests in this research obtained response times higher than similar scenarios in other executions. After assessing the obtained results, we found that one of the reasons for

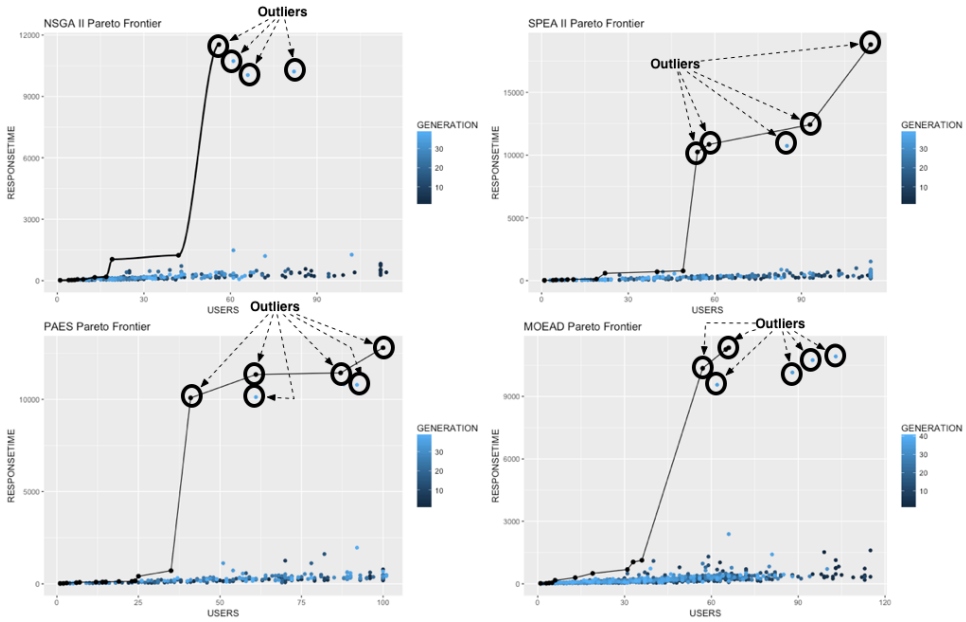


Figure B.1: Results obtained without Noise Reduction

higher response time is the occurrence of disk pagination. In order to more accurately test the response times of the outliers, we put forward an adaptation of the SEDR algorithm. Fig. B.1 presents the results without a Noise Reduction algorithm.

SEDR noise reduction is used by this research in multi-objective scenarios experiments where the objective of the experiments besides finding the tests with the longest response time also needs to find the Pareto frontier of the application.

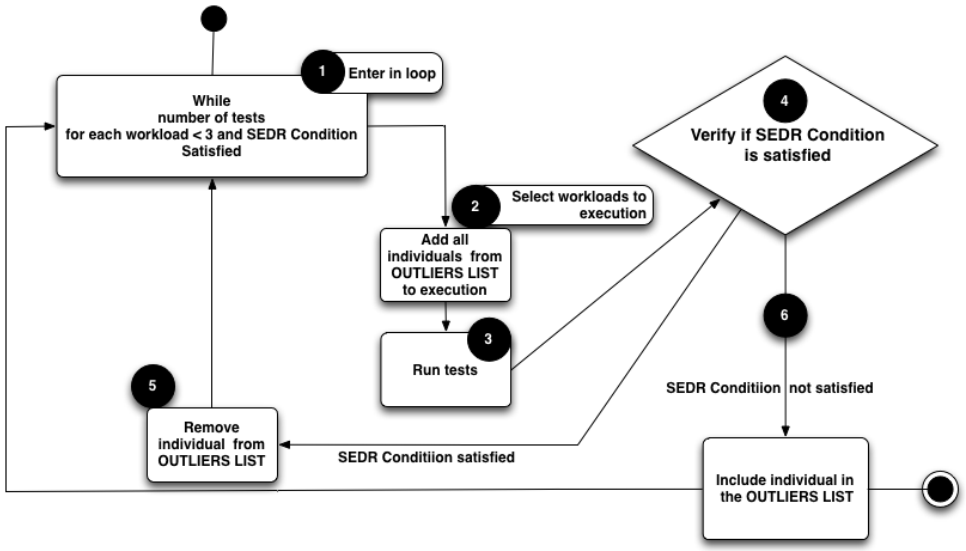


Figure B.2: SEDR customized algorithm

The experiments use a customized version of the SEDR algorithm for noise reduction. The algorithm runs as long as each workload has at least 3 samples or the SEDR condition is satisfied

for all workloads (Figure B.2 - ❶). The algorithm adds all workloads in OUTLIERS list to the list of execution (Figure B.2 - ❷). All workloads are performed and the SEDR condition is verified (Figure B.2 - ❸ and ❹). If the SEDR condition is satisfied the workload is removed from OUTLIERS list (Figure B.2 - ❺) otherwise the workload is included in the list if it is no longer present (Figure B.2 - ❻).

APPENDIX C – IADAPTER

IAdapter is a JMeter plugin designed to perform search-based stress tests. Fig. C.1 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 chosen by the test tool. In a test with JMeter, the tests scenarios are usually chosen by a test designer.

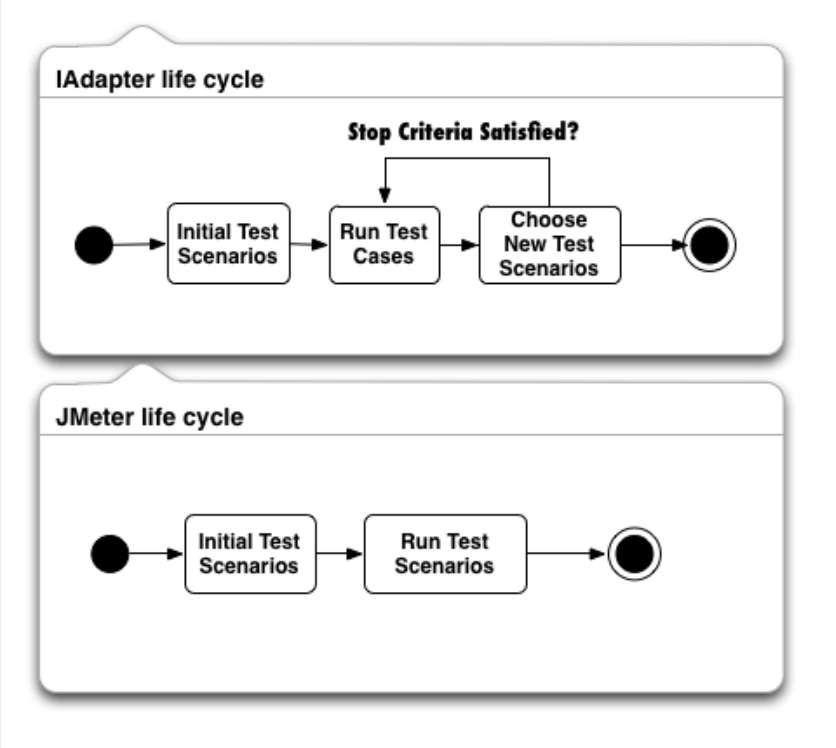


Figure C.1: IAdapter life cycle

C.1 IAdapter Visual Components

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

WorkLoadThreadGroup is a component that creates an initial population and configures the algorithms used in the IAdapter. Fig. C.2 presents the main screen of the WorkLoadThreadGroup component. The component has a name ❶, a set of configuration tabs ❷, a list of individuals by generation ❸, a button to generate an initial population ❹, and a button to export the results ❺. 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 require being logged in the application under test and bring the application back to its original state.

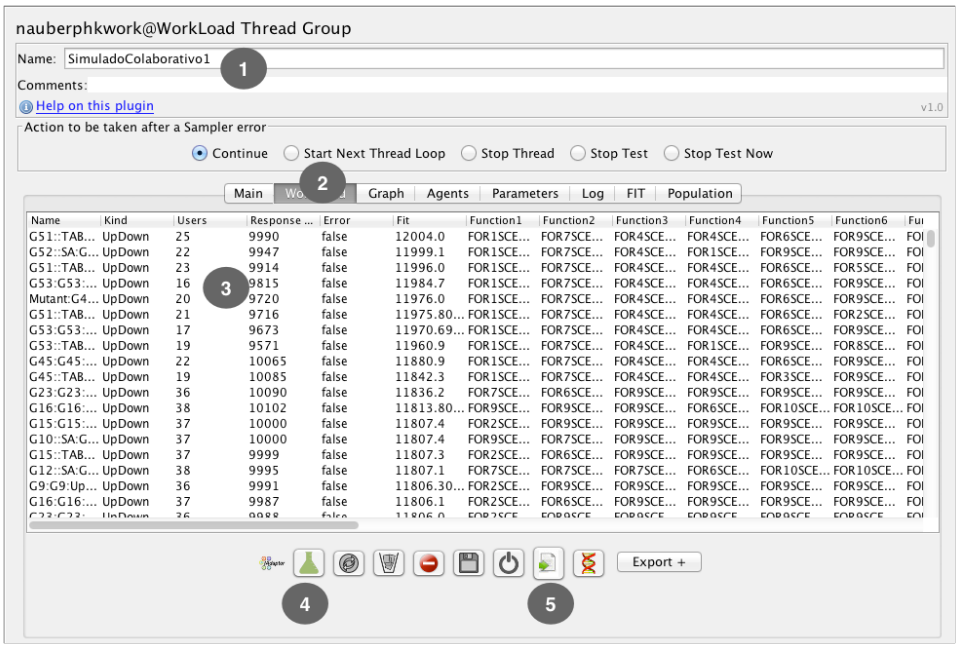


Figure C.2: WorkLoadThreadGroup component

C.2 The IAdapter architecture

In this section, We present the IAdapter main architecture. The testbed tool proposed consists of four main modules. Figure C.3 presents the main architecture of the solution proposed. The emulator module provides workloads to the Test module. The Test module uses a class loader to find all classes that extend AbstractAlgorithm in the classpath and run all workloads with each metaheuristic found. The Test Scenario library provides the scenario representation used by the metaheuristics and store the testbed results in a database. The Operation services are responsible for finding neighbors of some workload provided as a parameter and perform crossover operations.

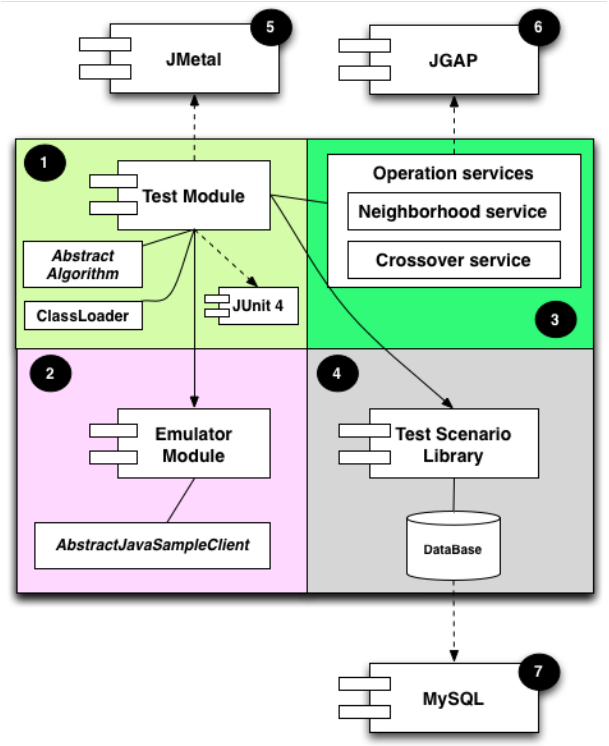


Figure C.3: IAdapter main architecture.

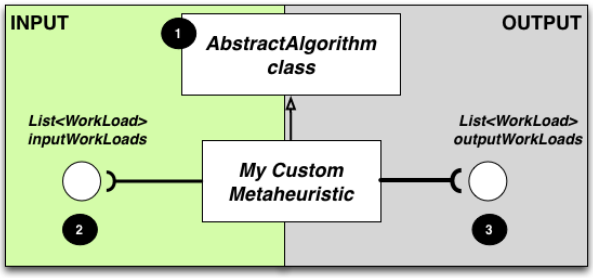
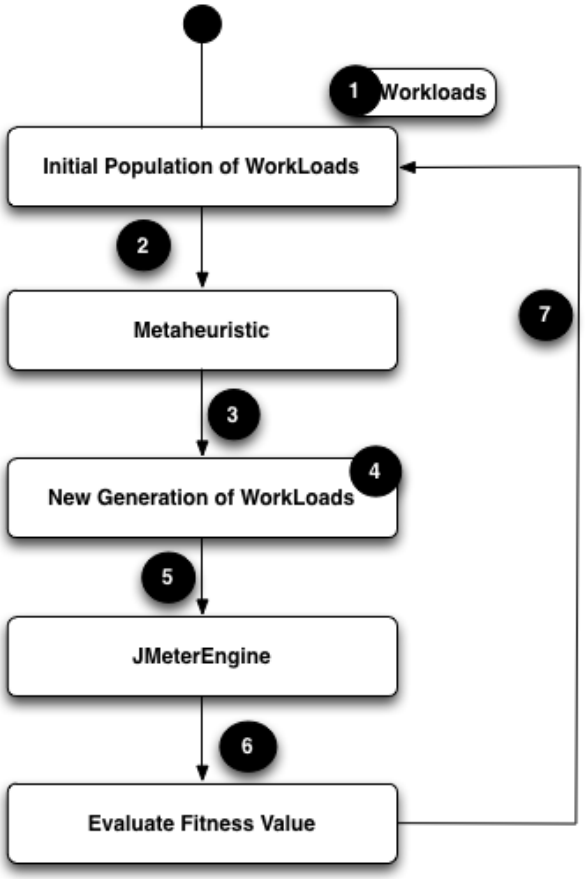


Figure C.4: Test Module class diagram.

C.2.1 Test Module

The Test Module (Figure C.3 -❶) is responsible for for the loading of all classes that extend AbstractAlgorithm in the classpath and perform the tests under the application. Figure C.4 shows the class diagram for custom and provided heuristics. All heuristic classes extend the class AbstractAlgorithm. The heuristics receives as input a list of workloads (Figure C.4 -❷) and must return a list of output workloads (the individuals selected for the next generation) (Figure C.4 -❸). Each workload represents an individual in the search space. There is an abstract class inherited from the AbstractAlgorithm named Multiobjective Algorithm, this class is used by multi-objective metaheuristics, which have a different execution flow in the system.

Figure. C.5 presents the Flowchart of the Test Module. Given an initial population (Figure C.5 -❶), a metaheuristic select a new set of workloads based on an objective function (Figure C.5 -❷). The chosen metaheuristic generate a new set of individuals based on crossover or neighborhood operators (Figure C.5 -❸). JMeterEngine run each workload (Figure C.5 -❺) and the chosen metaheuristic obtain a fitness value for each workload based on some objective function (Figure C.5 -❻). Each Metaheuristic could define your own objective function. After all these steps the cycle begins until the maximum number of generations it is reached (Figure C.5 -❼).



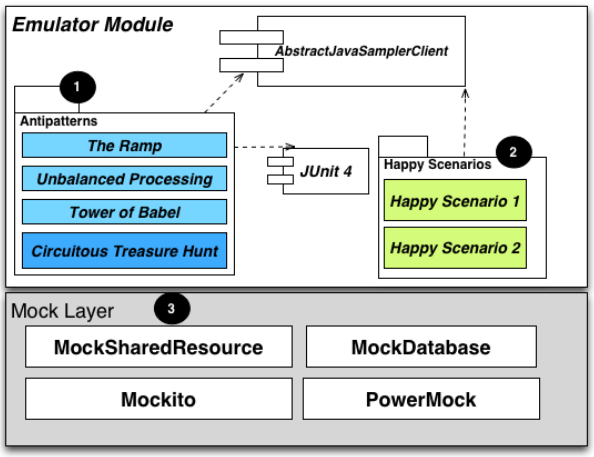


Figure C.6: Emulator module

Figure C.5: Flowchart of Test Module.

The WorkLoadThreadGroup class is responsible for start all threads that go simulate the users in the jmeter engine. Fig C.7 presents the WorkLoadThreadGroup life cycle. First, an instance of the class waits for the user to start the test (Stopped State - Fig. C.7 - ❶). Once the test is started, the start method is called (Fig. C.7 - ❷) and the instance goes to Running state (Fig. C.7 - ❸). After running all threads, the class goes to the finished state (Fig. C.7 - ❺).

Figure C.8 presents the start method sequence diagram. The WorkLoad Thread Group start method is responsible for load the multi-objective weights, synchronize all threads and get the new workloads from the database. Figure C.9 presents the threadFinish method, the method is triggered after the end of execution of each Thread. The method synchronizes all threads (Fig. C.9-❸), waiting for the execution of all instances to finish, stores the results, and starts new tests until the end of all generations (Fig. C.9-❹ and ❺).

C.2.2 Emulator Module

The Emulator Module is responsible for implementing and providing successful scenarios and the most common performance antipatterns (Figure C.3 -❷). All classes must extend the

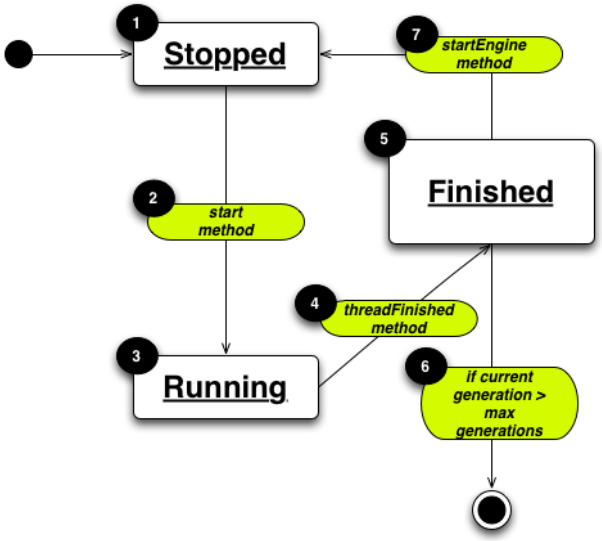


Figure C.7: WorkLoadThreadGroup class life cycle.

AbstractJavaSamplerClient class or use JUnit 4. The AbstractJavaSamplerClient class allows the creation of a JMeter Java Request. Figure C.6 presents the main features of the emulator module. The module implements 2 happy scenarios (Figure C.6 -❷) and 4 antipatterns test scenarios (Figure C.6 -❶), in its first version. The Mock Layer provides emulated databases and components for the test scenarios. The Mock Layer use the Mockito and PowerMocks frameworks (Figure C.6 -❸).

C.2.3 Test Scenario library

The representation of each individual is encapsulated in the WorkLoad class (Fig C.10). All test scenarios are referenced by string objects named *function1, function2, ..., function8, function9, and function10*.

C.2.4 External Dependencies

This section brieflys present the two main external dependencies of the IAdapter: jMetal and JGAP.

jMetal

jMetal, an object-oriented Java-based framework aimed at facilitating the development of metaheuristics for solving multi-objective optimization problems. jMetal provides a rich set of classes which can be used as the building blocks of multi-objective metaheuristics; thus, taking

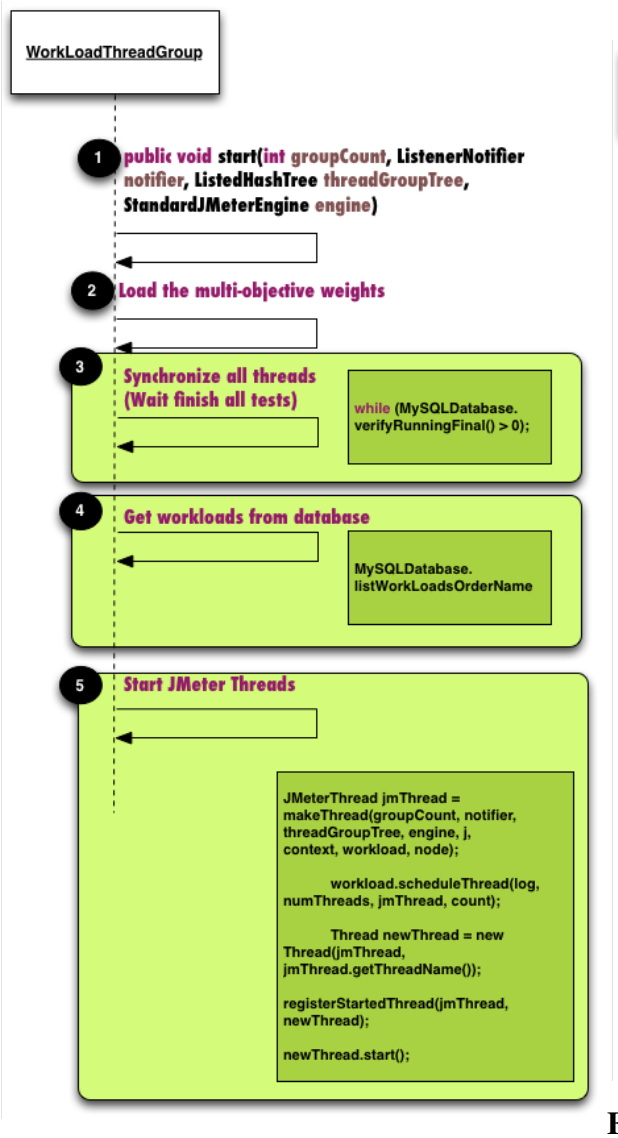


Figure C.8: WorkLoadThreadGroup start method.

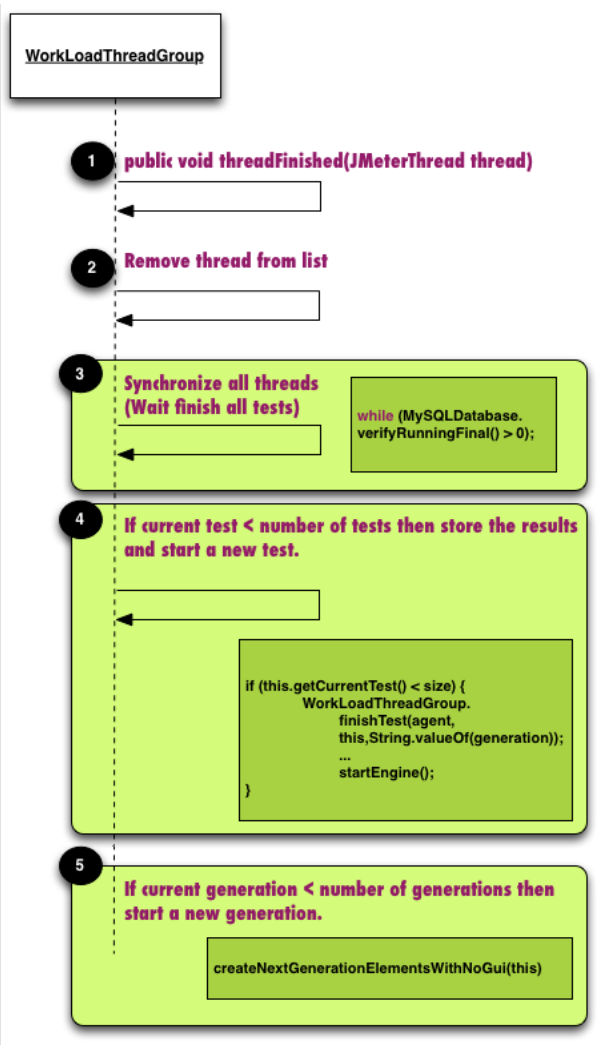


Figure C.9: WorkLoadThreadGroup threadFinished method.

advantage of code-reusing. The framework also incorporates a significant set of problems used as a benchmark in many comparative studies (DURILLO et al., 2006).

JGAP

JGAP is a GA package written in Java and distributed freely. Its design is modular, it is well documented and easily extensible, and active developer and user communities promote continual improvements. These characteristics support our goals of portability and flexibility. JGap allows in quite nice object way to set lots of genetic properties and use different population transformations. During the JGAP evolution process, chromosomes are exposed to multiple genetic operators that represent mating, mutation, etc. and then are chosen for the next generation

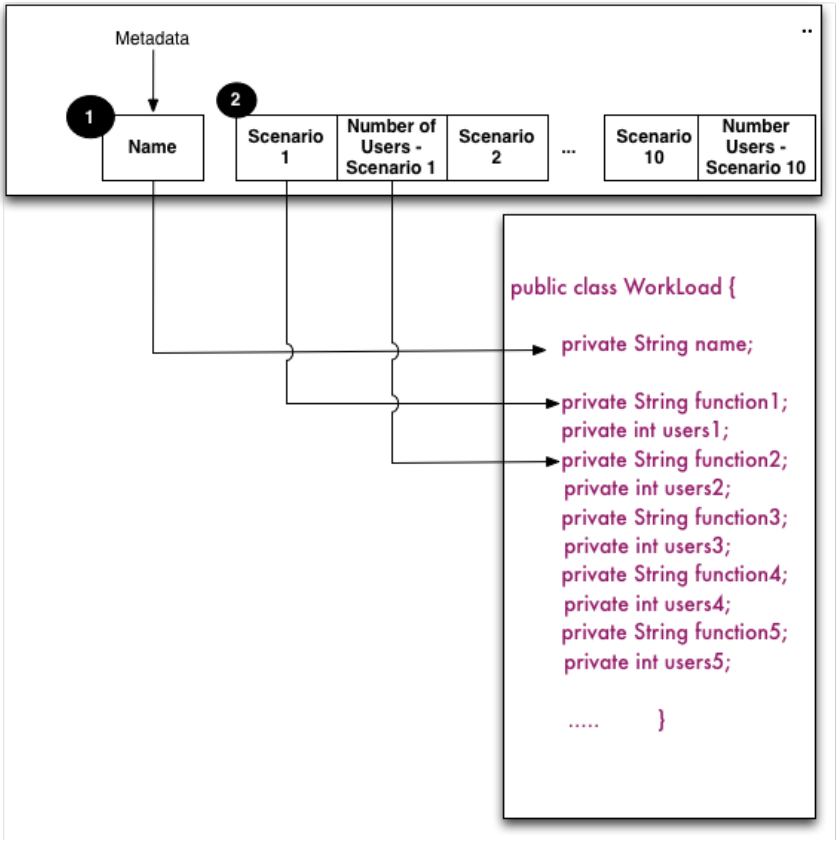


Figure C.10: WorkLoad class

during a natural selection phase based upon their fitness which is a measure of how optimal that solution is relative to other potential solutions. JGAP lets us choose what gene class to use to represent each gene in the chromosome. That provides the most flexibility and convenience (FIEBRINK; MCKAY; FUJINAGA, 2005).