# Search-Based Stress Testing with Multiobjective Evolutionary Algorithms: A Comparative Case Study

Nauber Gois[a], Pedro Porfírio[a], André Coelho[a]

[a] *Universidadede Fortaleza. Av. Washington Soares, 1321 - Edson Queiroz, Fortaleza - CE, 60811-905*

**Abstract**

Some software systems must respond to thousands or millions of concurrent requests. 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. 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. This paper investigates the use of the multi-objective NSGA-II, SPEA2, PAES and MOEA/D algorithms in search-based stress testing. MOEA/D metaheuristics obtained the best hypervolume value when compared with other approaches.

*Keywords:* Search-based Stress Testing, Multi-objective algorithms, Elsevier, template

*2010 MSC:* 00-01, 99-00

*Corresponding author

*Email address:* support@elsevier.com (André Coelho)

*URL:* www.elsevier.com (Pedro Porfírio)

[1] Since 1880.

## 1. Introduction

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 [1] [2] [3].

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 [4].

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 [5]. 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 [6]. Manual analysis of load testing is inefficient and error prone due to incomplete knowledge of test analyst about the application under test[7].

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 [4].

Search-based stress testing (SBST) is regarded as a promising approach to verify timing constraints [4]. A common objective of a load search-based test is to find scenarios that produce execution times that violate the specified timing constraints [8].

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.

The present study extends the article "Improving stress search based testing using a hybrid metaheuristic approach" [9] in order to ascertain if the use of multi-objective algorithms in search-based stress testing. This paper addresses the following research question:

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

One experiment was conducted to validate the proposed approach. The experiment was performed using an installed JPetStore application. The experiment presents the experimental results to compare four multi-objective algorithms in search-based stress testing.

The remainder of the paper is organized as follows. The next section gives background on multi-objective metaheuristics. Section 3 presents presents the Hybrid approach proposed by Gois et al. [9]. Section 4 presents the comparative experiment performed. Conclusions and further work are presented in Section 5.

## 2. Multi-objective optimization using evolutionary algorithms

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 [10]. 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. 1. 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.

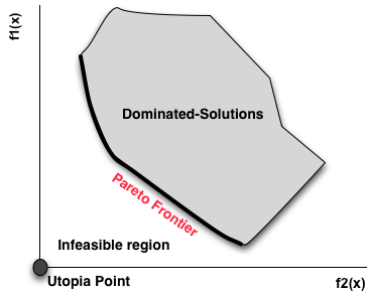


**Figure 1:** An optimized Pareto front example

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 [11].

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 [11].

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 [11].

### *2.1. NSGA-II: Nondominated Sorting Genetic Algorithm II*

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 [12]. 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- ❶ and ❷). The populations are ranked using multiple tournament selections, which consist of comparing two solutions (Figure 2- 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- 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- 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 [12]. 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 [13].

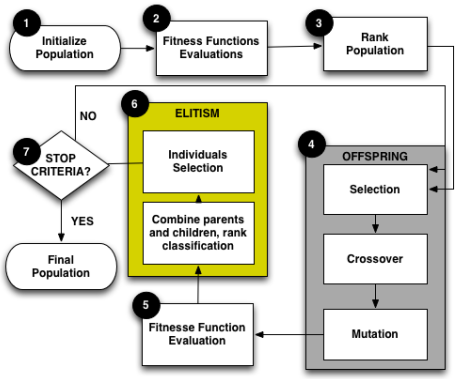


**Figure 2:** NSGA-II Algorithm

### *2.2. SPEA2: 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 in the archive assign a strength value S(i) $\in$ [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. 1. 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 [11] [14] [15].

**Algorithm 1** SPEA2 Algorithm [11]

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

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 3) [14] [16].

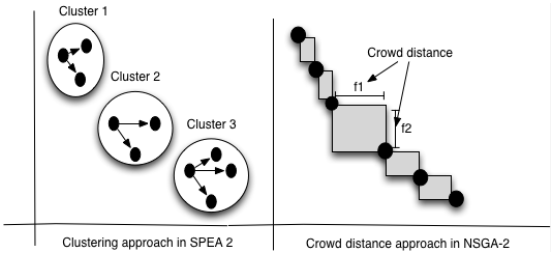


**Figure 3:** Comparison between SPEA-2 and NSGA-II [16]

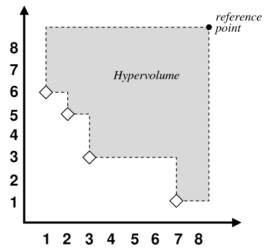


**Figure 4:** Hypervolume metric [17]

### *2.3. PAES: 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 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 [18]. 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 [18][19]. The algorithmic framework of PAES is described in Alg. 2.

### *2.4. MOEA/D: 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 [20]. 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,

**Algorithm 2** PAES Algorithm [18][19]

```
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
```

which makes MOEA/D have lower computational complexity at each generation than MOGLS and non-dominated sorting genetic algorithm II (NSGA-II) [21].

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 [21] [22]. Fig. 5 MOEA/D create a set of subproblems $W_i$. Each subproblem $W_i$ is improved to

obtain the Pareto frontier [22].

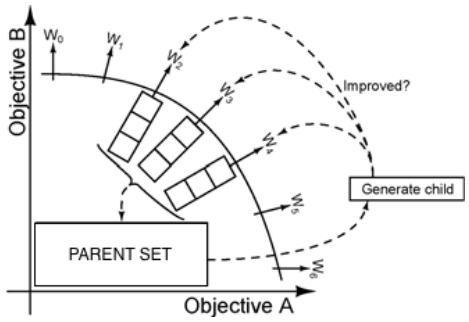


**Figure 5:** MOEA/D algorithm subproblems [22]

The Alg. 3 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 [23].

### *2.5. Comparing Multi-Objective Metaheuristics*

Deb states that are two orthogonal goals for any multi-objective algorithm [24]:

- 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 [25]. 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 [26]. 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 4 ) [25]. The reference point is found by constructing a vector of worst objects fitness value.

**Algorithm 3** MOEA/D Algorithm

**Initialization**
Generate initial population by uniformly spreading and randomly sampling from search space
Calculate the reference point for the Tchebycheff approach.
Evaluate Objective Values
Selection using tournament selection method
Selection of mating and updating range
Reproduction
Repair
Update of solutions
**Update**
While (not equal to termination condition)
Evaluate Objective Values
Selection using tournament selection method
Selection of mating and update range
Reproduction
Repair - if the searching element is out of boundary
Update the solutions
**Stopping Criteria**
**if** generation is a multiplication of a pre-set value of x **then**
    Update utility function;
**end if**

### *2.6. Noise Reduction*

Software is pervasive, which raises the value of testing it [27]. 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 noise 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 [28].

Standard Error Dynamic Resampling (SEDR), the 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 fall below a chosen threshold value [29]. It was proposed in [30] 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. 4.

**Algorithm 4** SEDR algorithm [29]

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

## 3. Improving Stress Search Based Testing using Hybrid Metaheuristic Approach

This section presents the Hybrid approach proposed by Gois et al. [9]. The solution proposed by Gois et al. makes it possible to create a model that evolves during the test. A plugin called iadapter was implemented for the research. IAdapter is a JMeter plugin designed to perform search-based stress tests. The plugin is available at `www.github.com/naubergois/newiadapter`.

The proposed solution model uses genetic algorithms, tabu search, and simulated annealing in two different approaches. The study initially investigated the use of these three algorithms. Subsequently, the study will focus on other population-based and

single point search metaheuristics. The first approach uses the three algorithms independently, and the second approach uses the three algorithms collaboratively (hybrid metaheuristic approach).

In the first approach , the algorithms do not share their best individuals among themselves. Each algorithm evolves in a separate way (Fig. 6). The second approach uses the algorithms in a collaborative mode (hybrid metaheuristic). In this approach, the three algorithms share their best individuals found (Fig. 7). The next subsections present details about the used metaheuristic algorithms (Representation, initial population and fitness function).

### *3.1. Representation*

The solution representation provides a common representation for all workloads. Each workload is composed by a linear vector with 21 positions (Figure 8 -❶). The first position represents an metadata with the name of an individual. The next positions represent 10 scenarios and their numbers of users (Figure 8 -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. 8 presents the solution representation and an example using the crossover operation. In the example, solution 1 (Figure 8 -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 8 -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 8 -❺). Figure. 8 -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.

### *3.2. 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

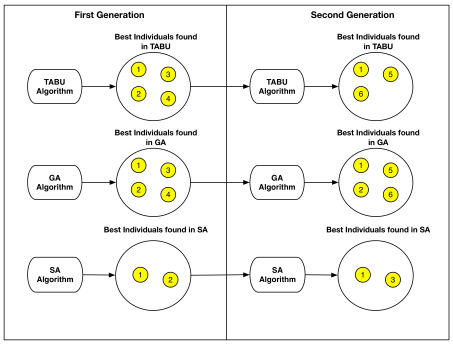


**Figure 6:** Use of the algorithms independently [9]

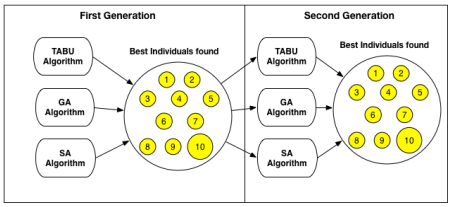


**Figure 7:** Use of the algorithms collaboratively [9]

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.

### *3.3. 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

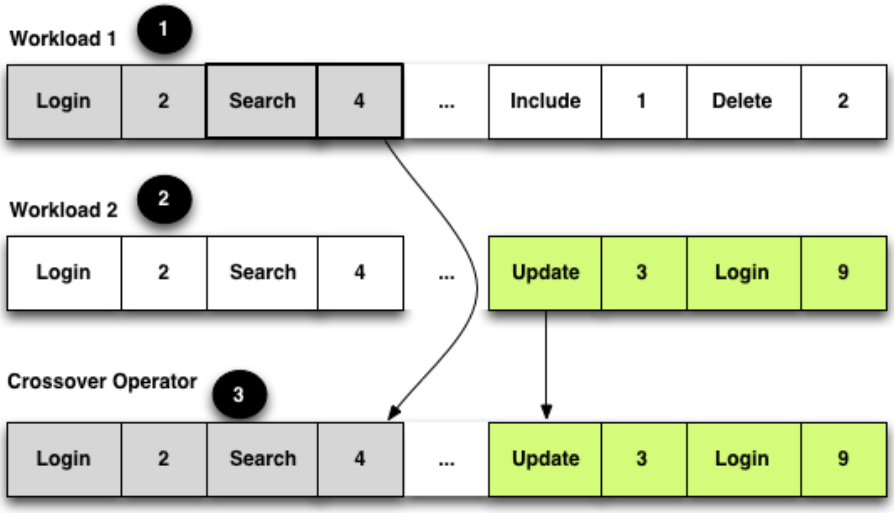


**Figure 8:** Solution representation, crossover and neighborhood operators [9]

applied to the IAdapter solution is governed by the following equation:

$$\begin{aligned} fit = {} & numberOfUsersWeight * numberOfUsers \\ & -90percentileweight * 90percentiletime \\ & -80percentileweight * 80percentiletime \\ & -70percentileweight * 70percentiletime \\ & -maxResponseWeight * maxResponseTime \\ & -penalty \end{aligned} \tag{1}$$

The users and response time factors were chosen because they are common units of measurement in load test tools [27]. The proposed solution's fitness function uses a series of manually adjustable user-defined weights (90percentileweight, 80percentileweight, 70percentileweight, maxResponseWeight, and numberOfUsersWeight). These weights make it possible to customize the search plugin's functionality. A penalty is applied when the response time of an application under test runs longer that the service level.

The penalty is calculated by the follow equation:

$$penalty = 100 * \Delta$$
$$\Delta = (t_{CurrentResponseTime} - t_{MaximumResponseTimeExpected}) \quad (2)$$

## 4. Perfomance Comparison

This section presents experiments to assert the benefits of multiobjective metaheuristics in search-based stress testing. 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. 9 presents the flowchart of Multiobjective Algorithms Adaptation. Given an initial population (Figure 9 -❶), the multiobjective algorithm implementation receives a set of workloads and generates a new set of individuals (Figure 9 -❷). JMeterEngine runs each workload (Figure 9 -❸) and the multiobjective algorithm ranks and classifies each workload based on the objective functions (Figure 9 -❹). After all these steps the cycle begins until the maximum number of generations is reached.

One experiments were performed to evaluate the use of multi-objective metaheuristics in search-based stress testing. The experiments use a SEDR adaptation algorithm. 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.

### *4.1. Experiment Research Questions*

The following research question is addressed:

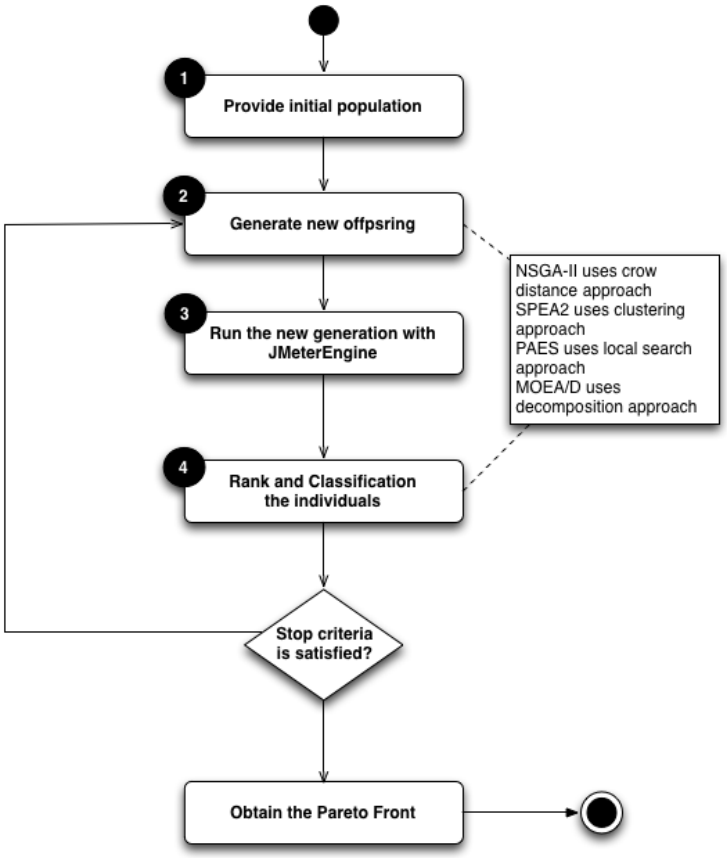


**Figure 9:** Flowchart of implemented algorithms

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

### *4.2. Variables*

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

### *4.3. 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 the 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.

### *4.4. Experiment Results*

Table 2 shows the hypervolume value of each algorithm. Dimensionality shows the number of dimensions used in the calculation of hypervolume. 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. Table 1 presents the test scenarios found in the Pareto frontier of MOEA/D.

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 10, 11, 12 and 13 present the Pareto frontier for each algorithm. Fig. 14 presents a comparative graph between all the algorithms.

**Table 1:** Response time and number of users in optimal scenarios found with MOEA/D

| N. | USERS | RESP TIME | Dogs Users | Enter Catalog Users | Fish Users | Register Users | Add Rem. Cart | Cart Users |
|---|---|---|---|---|---|---|---|---|
| ❶ | 1 | 18 | | 1 | | | | |
| ❷ | 5 | 35 | 1 | | | 3 | | 1 |
| ❸ | 6 | 50 | 2 | | | 3 | | 1 |
| ❹ | 7 | 320 | | | | 3 | | 4 |
| ❺ | 12 | 410 | | | | 3 | 5 | 4 |
| ❻ | 22 | 788 | | | 7 | 7 | 3 | 5 |
| ❼ | 72 | 1074 | 17 | 19 | 14 | 11 | | 11 |
| ❽ | 89 | 1396 | 8 | 7 | 10 | | 29 | 35 |

**Table 2:** Hypervolume by algorithm

| 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 |

### *4.5. 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.

### *4.6. 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.

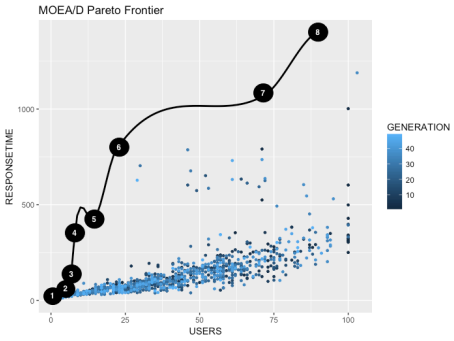


**Figure 10:** MOEA/D Pareto Frontier

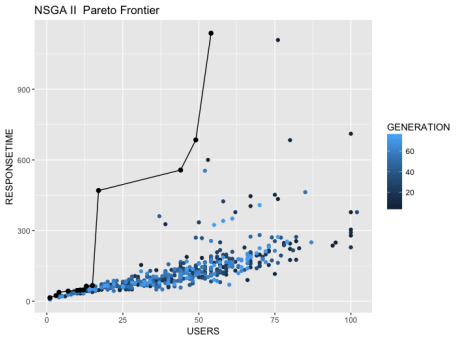


**Figure 11:** NSGA II Pareto Frontier

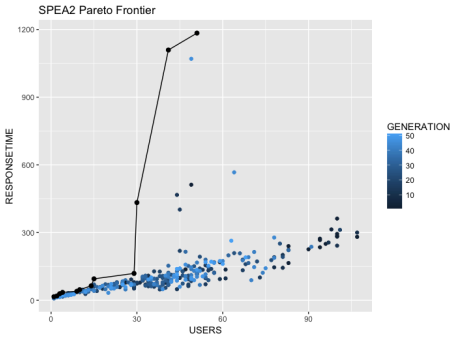


**Figure 12:** SPEA2 Pareto Frontier

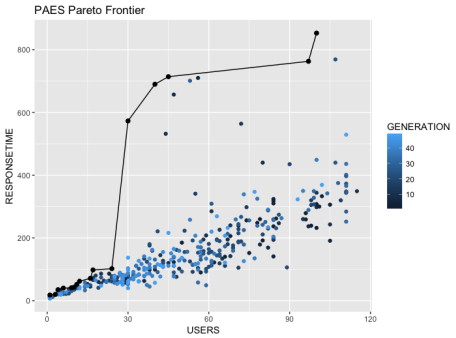


**Figure 13:** PAES Pareto Frontier

## 5. Conclusion

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

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 experiments found scenarios that contribute to the definition of the service level. 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

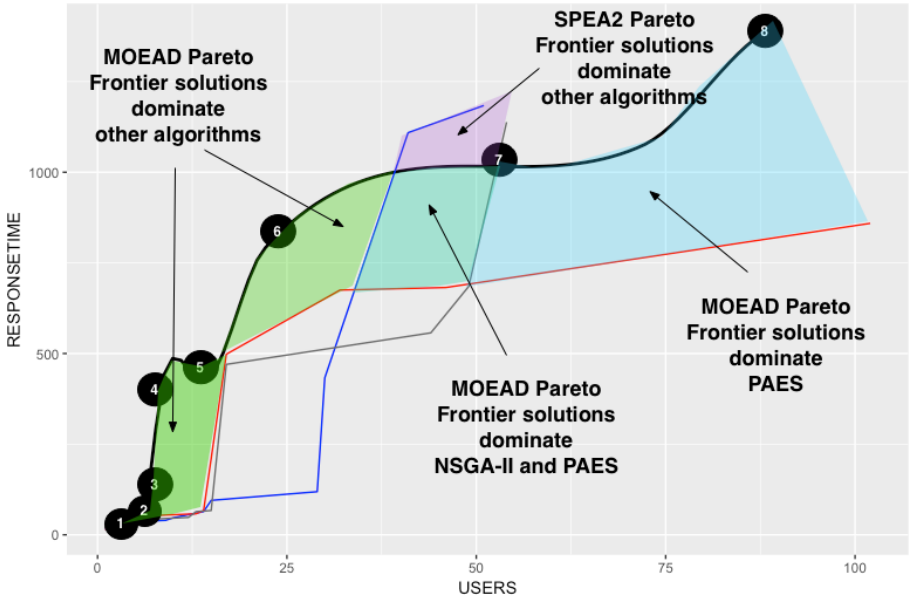


**Figure 14:** Comparative graph between MOEA/D, NSGA-II, SPEA2 and PAES Pareto Frontiers

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. 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. The choice of fitness function can be a difficult step in Genetic Algorithms. The fitness function is still manually adjusted and additional studies need to be performed to get the best weight values for the fitness function. 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.

## References

[1] Z. Jiang, Automated analysis of load testing results, Ph.D. thesis (2010).
URL http://dl.acm.org/citation.cfm?id=1831726

[2] I. Molyneaux, The Art of Application Performance Testing: Help for Programmers and Quality Assurance, 1st Edition, "O'Reilly Media, Inc.", 2009.

[3] A. Wert, M. Oehler, C. Heger, R. Farahbod, Automatic detection of performance anti-patterns in inter-component communications, 2014, pp. 3–12. doi:10.1145/2602576.2602579.
URL http://dx.doi.org/10.1145/2602576.2602579

[4] W. Afzal, R. Torkar, R. Feldt, A systematic review of search-based testing for non-functional system properties, Vol. 51, Elsevier B.V., 2009, pp. 957–976. doi:10.1016/j.infsof.2008.12.005.

[5] C. Trubiani, PhD Thesis in Computer Science Automated generation of architectural feedback from software performance analysis results Catia Trubiani, Language.

[6] M. Grechanik, C. Fu, Q. Xie, Automatically finding performance problems with feedback-directed learning software testing, Ieee, 2012, pp. 156–166. doi:10.1109/ICSE.2012.6227197.

[7] M. Arslan, U. Qamar, S. Hassan, S. Ayub, 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) 2015-Novem (September) (2015) 140–144. doi:10.1109/ICSESS.2015.7339023.
URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84958250651{&}partnerID=tZOtx3y1

[8] M. O. Sullivan, S. Vössner, J. Wegener, D.-b. Ag, Testing Temporal Correctness of Real-Time Systems — A New Approach Using Genetic Algorithms and Cluster Analysis —, 1998, pp. 1–20.

[9] N. Gois, P. Porfirio, A. Coelho, T. Barbosa, Improving Stress Search Based Testing using a Hybrid Metaheuristic Approach, in: Proceedings of the 2016 Latin American Computing Conference (CLEI), 2016, pp. 718–728.

[10] M. Harman, P. McMinn, A theoretical and empirical study of search-based testing: Local, global, and hybrid search, Vol. 36, 2010, pp. 226–247. doi:10.1109/TSE.2009.71.

[11] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the Strength Pareto Evolutionary Algorithm, Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems (2001) 95–100arXiv:arXiv:1011.1669v3, doi:10.1.1.28.7571.

[12] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II, Parallel Problem Solving from Nature PPSN VI (2000) 849–858doi:10.1007/3-540-45356-3_83.
URL http://repository.ias.ac.in/83498/

[13] S. Yoo, M. Harman, Pareto efficient multi-objective test case selection, Proceedings of the 2007 international symposium on Software testing and analysis - ISSTA '07 (2007) 140doi:10.1145/1273463.1273483.
URL http://doi.acm.org/10.1145/1273463.1273483{%}5Cnhttp://dl.acm.org/ft{_}gateway.cfm?id=1273483{&}type=pdf{%}5Cnhttp://portal.acm.org/citation.cfm?doid=1273463.1273483

[14] T. Tervonen, U. Kingdom, 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 (April). doi:10.1016/j.omega.2016.07.003.

[15] R. A. Matnei Filho, S. R. Vergilio, A multi-objective test data generation approach for mutation testing of feature models, Journal of Software Engineering Research and Development 4 (1) (2016) 4. doi:10.1186/s40411-016-0030-9.
URL http://jserd.springeropen.com/articles/10.1186/s40411-016-0030-9

[16] K. Deb, M. Mohan, S. Mishra, Evaluating the epislon-domination based multiobjective evolutionary algorithm for a quick computation of Pareto-optimal

solutions, Evolutionary Computation Journal 13 (4) (2005) 501–525. doi:10.1162/106365605774666895.

[17] R. Lacour, K. Klamroth, C. M. Fonseca, A box decomposition algorithm to compute the hypervolume indicator, Computers and Operations Research (2015) 1–21arXiv:1510.01963, doi:10.1016/j.cor.2016.06.021.

[18] J. Knowles, D. Corne, 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) 1 (1999) 98–105. doi:10.1109/CEC.1999.781913.

[19] M. Oltean, A. Abraham, K. Mario, Multiobjective Optimization Using Adaptive Pareto Archived Evolution Strategy (2005) 1–6.

[20] K. Michalak, The effects of asymmetric neighborhood assignment in the MOEA/D algorithm, Applied Soft Computing Journal 25 (2014) 97–106. doi:10.1016/j.asoc.2014.07.029.
URL http://dx.doi.org/10.1016/j.asoc.2014.07.029

[21] Q. Zhang, H. Li, MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition, IEEE Transactions on Evolutionary Computation 11 (6) (2007) 712–731. doi:10.1109/TEVC.2007.892759.

[22] T. McConaghy, P. Palmers, M. Steyaert, G. G. E. Gielen, Trustworthy genetic programming-based synthesis of analog circuit topologies using hierarchical domain-specific building blocks, IEEE Transactions on Evolutionary Computation 15 (4) (2011) 557–570. doi:10.1109/TEVC.2010.2093581.

[23] T. J. Yuen, R. Ramli, Comparison of Computational Efficiency of Moea \ D and Nsga-Ii for Passive Vehicle Suspension Optimization 2 (Cd).

[24] K. Deb, Multi-objective optimization using evolutionary algorithms, Vol. 16, John Wiley & Sons, 2001.

[25] G. Janssens, J. Pangilinan, Multiple criteria performance analysis of non-dominated sets obtained by multi-objective evolutionary algorithms for optimisation, Artificial Intelligence Applications and Innovations (2010) 94–103.

[26] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study\nand the strength Pareto approach, IEEE Transactions on Evolutionary Computation 3 (4) (1999) 257–271. doi:10.1109/4235.797969.

[27] C. Sandler, T. Badgett, T. Thomas, The Art of Software Testing, John Wiley & Sons, 2004, p. 200.

[28] P. Rakshit, A. Konar, S. Das, Noisy evolutionary optimization algorithms – A comprehensive survey, Swarm and Evolutionary Computation 33 (2017) 18–45. doi:10.1016/j.swevo.2016.09.002.
URL http://dx.doi.org/10.1016/j.swevo.2016.09.002

[29] F. Siegmund, A. H. C. Ng, K. Deb, A comparative study of dynamic resampling strategies for guided Evolutionary Multi-objective Optimization, 2013 IEEE Congress on Evolutionary Computation, CEC 2013 (2013008) (2013) 1826–1835. doi:10.1109/CEC.2013.6557782.

[30] A. Di Pietro, L. While, L. Barone, Applying evolutionary algorithms to problems with noisy, time-consuming fitness functions, Proceedings of the 2004 Congress on Evolutionary Computation (2004) 1254–1261doi:10.1109/CEC.2004.1331041.
URL http://ieeexplore.ieee.org/xpl/freeabs{_}all.jsp?arnumber=1331041