A Systematic Mapping addressing hyper-heuristics within Search-Based Software Testing ☆

Juliana Marino Balera[1], Valdivino Alexandre de Santiago Júnior [1,2]
juliana.balera@inpe.br,valdivino.santiago@inpe.br

[1] *Instituto Nacional de Pesquisas Espaciais (INPE)*
Laboratório Associado de Computação e Matemática Aplicada (LABAC) - Av. dos Astronautas 1758, São José dos Campos, SP, Brazil, 12227-010
www.inpe.br
[2] *The University of Nottingham*
School of Computer Science - Jubilee Campus, Wollaton Road, Nottingham, United Kingdom, NG8 1BB
www.nottingham.ac.uk

Abstract

Context: Search-based Software Testing (SBST) is a research field where testing a software product is formulated as an optimization problem. It is an active sub-area of *Search-based Software Engineering* (SBSE) where many studies have been published and some reviews have been carried out. The majority of studies in SBST has been adopted meta-heuristics while hyper-heuristics have a long way to go. Moreover, there is still a lack of studies to perceive the state-of-the-art of the use of hyper-heuristics within SBST.

Objective: The objective of this work is to investigate the adoption of hyper-heuristics for Software Testing highlighting the current efforts and identifying new research directions.

Method: A *Systematic Mapping* study was carried out with 5 research questions considering papers published up to may/2019, and 4 different bases. The research questions aims to find out, among other things, what are the hyper-heuristics used in the context of Software Testing, for what problems hyper-heuristics have been applied, and what are the objective functions in the scope of Software Testing.

☆Fully documented templates are available in the elsarticle package on CTAN.

Results: A total of 734 studies were found via the search strings and 164 articles were related to Software Testing. However, from these, only 26 papers were actually in accordance with the scope of this research and 3 more papers were considered due to snowballing or expert's suggestion, totalizing 29 selected papers. Few different problems and application domains where hyper-heuristics have been considered were identified.

Conclusion: Differently from other communities (Operational Research, Artificial Intelligence), SBST has little explored the benefits of hyper-heuristics which include generalization and less difficulty in parameterization. Hence, it is important to further investigate this area in order to alleviate the effort of practitioners to use such an approach in their testing activities.

Keywords: Search-Based Software Testing, Hyper-heuristics, Systematic Mapping, Evolutionary Algorithms, Genetic Algorithms, Meta-heuristics

1. Introduction

Search-Based Software Testing (SBST) is a sub-area of *Search-Based Software Engineering* (SBSE) which has been receiving a lot of attention from the academic community [1]. By formulating the testing of a software product as an optimization problem, SBST can benefit of several solutions which has long been used in Operational Research, Artificial Intelligence, and similar areas.

Meta-heuristics such as Evolutionary Algorithms (Genetic Algorithm [2, 3]), *Particle Swarm Optimization* (PSO) [4, 5], Simulated Annealing [6, 7], Tabu Search [8, 9] have been considered for Software Testing. Although meta-heuristics have long been proved beneficial to solve real-world complex search problems, in addition to the ones related to Software Testing, such as scheduling, clustering, educational timetabling, and space allocation, it is still not so easy to apply such meta-heuristics to new optimization problems, or even new instances of similar problems [10]. Some of the reasons of such difficulties are the usual high number of parameters or algorithm choices the practitioner must define, the absence of proper guidelines to select them, and the lack of general-

ization in applying meta-heuristics.

In this context, *hyper-heuristics* [10, 11] have been showing to be a promising alternative compared with common heuristics or meta-heuristics, precisely because it overcomes the obstacles previously mentioned. According to Burke et al. (2013) [10], a hyper-heuristic is "a search method or learning mechanism for selecting or generating heuristics to solve computational search problems". Hence, in hyper-heuristics, the search is performed in a search space of heuristics (or heuristics components) instead of being performed directly in the decision space (space of solutions). Hyper-heuristics have been used for solving several types of problems such as scheduling [12], vehicle routing [13] and satisfiability [14], just to name a few.

SBST community has been gradually shifting to approaches based on hyper-heuristic as Cohen recently stated [15]. Problems such as *Combinatorial Interaction Testing* (CIT) data generation [16] and *Integration and Test Order* (ITO) [17] have been addressed, but there is no indication in the literature of how wide and in how many ways hyper-heuristics have been considered. In other words, some open research questions are:

- What are the hyper-heuristics used in the context of SBST? Are there more generation or selection hyper-heuristics?
- What are the problems addressed by such strategies? For instance, they have been used only for test case/data generation or they have been considered for other problems?
- What are the objective functions the studies addressed? What has been prevailed: single or multi-objective problems?
- What is the validation context related to the application of hyper-heuristics?
- Which artifacts (source code, models, etc.) they have been relied on?

Several secondary studies (*Systematic Literature Reviews* (SLR), surveys) have been published within the search-based context such as [18, 19, 20, 21,

22, 23, 24]. However, these studies had a broader scope regarding the strategies for SBSE/SBST while we are interested in hyper-heuristics for Software Testing. Hence, none of the previous secondary studies addressed specifically hyper-heuristics within SBST with the research questions previously described.

Note that hyper-heuristics can also be considered an Artificial Intelligence technique and it is a well known fact that it has played an increasingly important role in several disciplines, including Software Engineering and, in particular, in Software Testing. Thus, it is important to know the state-of-the-art regarding the development and application of hyper-heuristics for Software Testing. This is then the objective of this paper where was have conducted a *systematic mapping* study considering papers up to may/2019 (in other words, we have covered all the papers up to may/2019) and 4 different bases. A total of 734 studies were found via the search strings and 164 articles were related to Software Testing. However, from these, only 26 papers were actually in accordance with the scope of this research and 3 more papers were considered due to snowballing or expert's suggestion, resulting in 29 selected papers. Was identified few different problems and application domains where hyper-heuristics have been considered.

This paper is organized as follows. Section 2 presents background and related work. In Section 3, the research methodology to conduct the systematic mapping is shown. Results and the discussion related to the research questions are in Section 4. Suggestions of future directions within this research field are presented in Section 5. Finally, conclusions are shown in Section 6.

2. Background and related work

This section presents an overview of some fields that are important to this systematic mapping. It also discusses some secondaries studies that are closely related to this paper.

2.1. Search-Based Software Testing

In SBST, test objectives are materialized in the form of objective functions [25], and thus optimization algorithms can be used to solve problems such as

CIT case/data generation and ITO.

Currently, Software Engineering problems consider more than a single goal, since the software metrics required for validation and the quality of the solution are taken into account. Consequently, it may be necessary for the tester to deal with multiple and likely conflicting objectives (for example, to increase the fault-revealing potential of a set of test cases while decreasing the number of test cases, and thus minimizing the costs). Hence, SBST community has moved towards multi-objective problems [26, 27, 28, 29]. Coverage, feasibility, similarity, cost, execution time, diversity, and length distribution of test suites are some example of objective functions that have been tackled in multi-objective test case/data generation.

2.2. Hyper-heuristics

In accordance with the definition proposed by [10], as presented in Section 1, in a hyper-heuristic, the search is done in a heuristic search space (or heuristic components, such as search operators) instead of being made directly in the decision space (space of solutions). In this way, hyper-heuristic approaches solve the problem indirectly, through the selection/generation of *Low-level Heuristics* (LLH).

In [11], the authors presented a possible classification for hyper-heuristics [11, 10]. The authors divided the classification in two domains: *feedback* and *nature of the heuristic search space*. The first domain concerns the type of learning to be used, which can be *online*, where virtual information about the performance of LLHs is used to dynamically select them, and *offline*, which uses a set of training instances to gain knowledge. Moreover, *no-learning* means that no feedback information is used to guide the search process.

The second domain, *nature of the heuristic search space*, defines the characteristics of the search space, whether formed by the LLH set (that is, the search space of hyper-heuristics) or formed by the possible problem solutions (decision space). In the context of the search space of the LLH set, there may exist *selection heuristics*, which are methodologies designed to select from an

already existing LLH set, and *generation heuristics*, which are methodologies that generate new LLH from other pre-existing ones. Now, in the context of the decision space, that is where the selection or generation heuristics will act, it is possible to have: (i) *construction (constructive) heuristics*, consisting of methodologies that start from an incomplete solution and gradually construct a complete solution; and (ii) *pertubation (perturbative) heuristics*, which consist of methodologies that depart from a complete solution, gradually modifying its structure in order to produce better results in relation to the original.

2.3. Related work

This section has as main objective to address some of the secondary studies and reviews which have already been conducted within the search-based context.

In [18], the aim is to analyze 5 of the major SBSE techniques in the field of various areas of Software Engineering, including Software Testing, in the last decade. Such techniques are Simulated Anneling, Genetic Algorithms, *Ant Colony Optimization* (ACO), PSO and Tabu Search. It defined research questions which take into consideration the main areas of application of Software Engineering and the main contributors.

The study [19] analyzes the current state-of-the-art of experimental applications of SBST to *Model-based Testing* (MBT), presenting the limitations of current solutions and future directions. It defined research questions which take into consideration the current state-of-the-art of SBST in the context of MBT.

The main objective of the study [20] was to provide an overview of the research in the field of testing of competing programs, classifying them into various categories. The review focuses on applied techniques, the methodologies followed, and the tools used in these approaches. The research questions focused on the current state-of-the-art research in the area of competing program testing.

In [21], the authors explored the concepts of bio-inspired algorithms to structure Resilient Systems, Genetic Strategies in generating test data, ACO algorithms for analysis, *Artificial Immune System* (AIS) Mutation Testing, and fault tolerant approaches inspired by immunity principles to increase software

reliability. In this study, no research questions were defined.

The aim of the study [22] is to identify how SBST has been explored in the context of Mutation Testing, how objective functions are defined and the challenges and opportunities of research in the application of meta-heuristics as search techniques. The research questions defined in this paper explore the characteristics that describe the context of SBST applications for use in Mutation Testing.

In [23], the objective is to make a bibliographical review on the application of the algorithms of optimization based on ant colonies in the context of several levels of Software Engineering, including Software Testing.

The study [24] is a comprehensive review aimed at finding trends in the field of SBST by examining software testing methods and literature. Several points related to SBST are addressed, such as open problems and challenges, fundamental materials and methods, research techniques and future scope. Meta-heuristics like Hill Climbing, Simulated Anneling, Tabu Search, Genetic Algorithms, among others, are mentioned. But there is no specific mention of hyper-heuristic and how it can be used within Software Testing.

In [30], McMinn discusses possible future research areas and open issues in the context of SBST based on the analysis of previous studies and the current state-of-the-art (at that time). The author goes through points such as search-based optimization algorithms, meta-heuristics, and their possible applications. Again, there is no specific mention of hyper-heuristic and how it can be used within Software Testing.

Based on the above mentioned studies, it is possible to conclude that several efforts are aimed at investigating the techniques used in the context of SBST, some of them shows a validation of the application of hyper-heuristics. However, these studies investigate the various techniques used in the context of a specific testing task (narrow scope) but in a broader scope regarding the strategies, where meta-heuristics (mostly), greedy algorithms, and rarely hyper-heuristics are mentioned. But, a secondary study that investigates exclusively the development and use of hyper-heuristics in the context of SBST as not beed

conducted up to now. Since hyper-heuristics is an emerging area for Software Testing, thus this systematic mapping aims to give a contribution in this sense.

3. Research methodology

One of the objectives of this work is to provide resources for the contextualization of hyper-heuristics applied to Software Testing. For this it is necessary to conduct a secondary study. According to [31], the methods for the generation of secondary studies commonly used in Software Engineering are systematic mapping and SLR. An systematic mapping is a systematic method for defining a classification scheme in a field of interest by analyzing the frequency of publications for each of these categories. On the other hand, an SLR aims to analyze the existing primary studies and describes its methodology and results in depth.

This study is an systematic mapping because it deals with a broader research area (Software Testing activity as a whole) although in a narrow scope regarding the strategies to accomplish Software Testing (the focus is on hyper-heuristic). The option for an systematic mapping was motivated by the need to have a general idea of the research field (hyper-heuristic within SBST) since there is an absence of such a view.

3.1. Planning

The first step of the *planning* phase aims at ensuring the need of the systematic mapping. In order to determine whether this secondary study was necessary, a preliminary search was performed on the bases *ACM Digital Library*, *IEEEXplore*, *ScienceDirect*, and *Scopus* to realize whether other secondary studies (SLRs, systematic mappings, surveys) had already been conducted specifically addressing hyper-heuristics within SBST. Section 2.3 summarizes the main findings and, as already pointed out, none of the previous secondary studies exclusively investigated the use of hyper-heuristics in detail within SBST. The search string was defined as follows:

(search-based OR "search based" OR "software testing") AND (review OR "systematic literature review" OR "systematic mapping")

The *research questions*, RQ_x, are shown in Table 1 where the aims of such questions are presented as well. In order to define these questions, typical aspects related to search-based optimization in the context of hyper-heuristics (RQ_1, RQ_3), investigation of the practical application of hyper-heuristics for Software Testing (RQ_2, RQ_5), and elements of the software development lifecycle (RQ_4) were taken into account. Columns **id** and **property** (where each of the properties is represented by $Prop_x$) of Table 1 will be discussed later.

id	**research question**	**aim**	**id**	**property**
RQ_1	What are the hyper-heuristics used in the context of SBST? Do we have more generating or selection hyper-heuristics ?	identify the main hyper-heuristics used in the context of SBST	$Prop_1$	types of hyper-heuristics
RQ_2	What are the problems addressed by such strategies? For Instance, they have been used only for test case generation or they have been considered for other purposes?	identify the problems addressed by the strategies based on hyper-heuristics for software testing	$Prop_2$	software testing activity
RQ_3	What are the objective functions considered?	identify the main objectives needed to solve a given problem in the context of SBST	$Prop_3$	test objective
RQ_4	Which artifacts (source code, models, ...) they relied on?	identify the software artifacts that are the basis for the application of the hyper-heuristic approach	$Prop_4$	software artifacts
RQ_5	What is the validation context related to the application of hyper-heuristics for software testing?	identify the validation context (case studies, etc.) where hyper-heuristics were applied	$Prop_5$	validation context
			$Prop_{5.1}$	research method

Table 1: Research questions

Within the research protocol, 4 bases were selected: *ACM Digital Library*, *ScienceDirect*, *Scopus* and *IEEEXplore*. The search string defined for the first

3 bases are defined as follows:

(hyper-heuristic OR hyperheuristic OR hyper-heuristics OR hyperheuristics) AND ("search-based software testing" OR "search-based testing" OR "search based software testing" OR "search based testing" OR "software testing" OR search-based OR "search based")

Due to the peculiarities of the *IEEEXplore* database, its search string is different from the one previously defined and it is shown below:

(hyper-heuristic OR hyperheuristic OR hyper-heuristics OR hyperheuristics) AND ("search-based software testing" OR "search-based testing" OR "search based software testing" OR "search based testing")

In order to evaluate the quality of the search strings, an analysis of the references of the selected articles was accomplished: it was verified, in each of the studies, which of their references were also retrieved by the search string used. In other words, the articles retrieved and selected were used as a kind of "guide" that allows to verify if it was possible, by means of the search string, to return what is really related to the subject studied in this research. All articles selected had articles as references that were also retrieved by the defined search string.

The inclusion criteria are: (i) primary studies in conferences and journals; (ii) secondary studies in conference and journals; (iii) articles published until may/2019, and; (iv) papers written in English. The exclusion criteria are: (i) editorial, abstract, short or white paper, and; (ii) duplicated studies. Note that although one of the inclusion criteria was that papers must be written in English, it was selected one paper in Portuguese which was an expert's suggestion. This paper was the only one that used entire *Multi-Objective Evolutionary Algorithms* (MOEAs) as LLHs of a selection hyper-heuristic.

3.2. Conducting

The search encompassed published articles until may/2019. The search itself

began in december/2017 but the bases were systematically verified again up to may/2019.

Figure 1 shows the amount of resulting articles in each of the main steps of the *conducting* phase. The red boxes represent the number of articles removed, the boxes in green represent the number of articles included, and the boxes in gray represent the number of articles as the result of each step. Finally, the blue frame displays the number of selected articles, used for the data extraction.

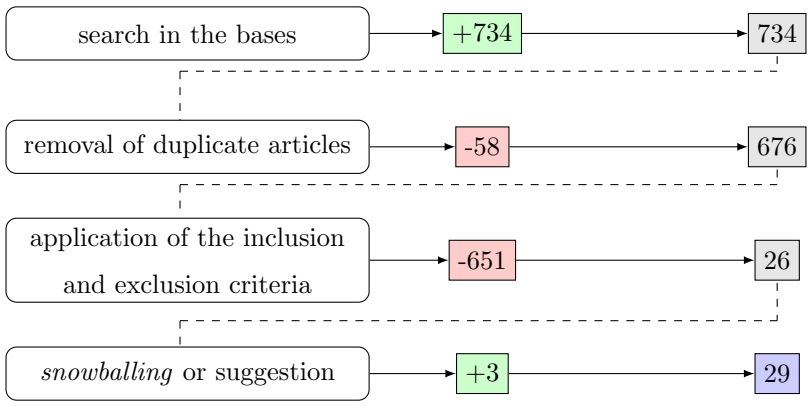

Figure 1: Number of papers returned in each main step of the *conducting* phase. Adapted from [32]

The last step of the *conducting* phase, data extraction, was done based on [33], which defined some essential properties that should be evaluated in each of the selected articles, in order to support the answer of the research questions. The properties are described in Table 1. Each of the properties described, $Prop_x$, is associated with one of the research questions. Now each of these properties is briefly described.

Types of Hyper-Heuristics ($Prop_1$)

This property is related to the types of hyper-heuristics, classified according to [11]: type (selection or generation heuristic), base algorithm, LLH set, learning method (*online*, *offline* or *no-learning*), and selection mechanism.

Software Testing Task ($Prop_2$)

This property highlights the Software Testing task involved in the application

of hyper-heuristics.

Test Objective ($Prop_3$)

This property highlights the test objectives, i.e. the objective functions, that were addressed by the hyper-heuristic approach, taking into account the type of test objective and the number of objectives evaluated by the hyper-heuristic.

Software Artifacts ($Prop_4$)

In this property, the aim is to know which are the related artifacts considered by the selected papers.

Validation Context ($Prop_5$)

This property characterizes the type of validation context used to evaluate the hyper-heuristic approach.

Research Method ($Prop_{5.1}$)

The studies can be classified according to the research methodologies too. As [33] proposed, these are the classes considered in this study: (i) case study; (ii) experiment; (iii) report; (iv) survey; (v) systematic mapping, and; (vi) SLR.

3.3. Threats to validity

3.3.1. Publication bias

In accordance with [34], threats to the publication bias consist in the consideration of elements that can make the developed research tendentious or not representative. In this direction, it is important to consider that during the development of a research, it may be that the researcher tends to emphasize the positive results in relation to the performance of the approach proposed by him, which means that the experimental results are not completely transparent.

According to the defined research questions for this systematic mapping context (see Table 1), which takes into account aspects such as the type of hyper-heuristic used, the case studies and the problem solved, the influence of the results achieved individually by each approach is minimal. For example, it is not relevant in the context of this work to take into account the performance of hyper-heuristics.

In addition, the systematic mapping was conducted in the most general manner possible in terms of publication vehicles and dates. In this way, the study was done in the most comprehensive way because it does not privilege certain publication vehicles.

3.3.2. Identification of primary studies

The quality of the identification of the primary studies can be influenced by the search strings defined in the research protocol, since the primary studies are obtained through the submission of these strings to the scientific bases. However, the poor definition of these search strings, i.e. the choice of terms not suitable to constitute them, can result in a search not sufficiently broad.

The technique used to evaluate the quality of the search strings used in this systematic mapping was made based on the data from Table 2. In order to evaluate the quality of the selected papers, it was took into account the citations of the articles: each of the bibliographic references of each of the selected papers was observed in order to verify if any relevant study of the area was not returned by the defined search string. The outcome is that all relevant studies cited by the selected articles were returned in this search.

Based on the analysis of Table 2, where each of the selected article is identified by P_x (more details about the papers are in Table 3), it is also possible to conclude that the article with the greatest influence in the context of this research is article P_{12}, which was cited by most other studies.

4. Results and analysis

Results and analysis of this systematic mapping are in this section. A total of 734 studies were found at first by means of the search strings. The vast majority of these studies were not related to the scope of this research. As can be seen in Figure 2, these 734 articles were grouped into 22 classes.

Out of the 164 articles related to Software Testing, only 29 were actually within the scope of this research, considering the 3 papers suggested by an expert, and these are listed in Table 3. The others were discarded because

article	**cites**
P_1 [16]	P_4, P_{12}
P_2 [17]	P_9, P_{11}, P_{12}
P_7 [15]	P_8
P_8 [35]	P_{12}
P_9 [36]	P_{12}
P_{10} [37]	P_3, P_9, P_{11}, P_{12}, P_{14}
P_{11} [38]	P_8, P_9, P_{12}
P_{13} [39]	P_3, P_9, P_{11}, P_{12}, P_{14}
P_{14} [40]	P_3, P_9, P_{11}, P_{12}
P_{15} [41]	P_2, P_9, P_{16}, P_{17}
P_{16} [42]	P_9, P_{12}
P_{18} [43]	P_9, P_{12}
P_{22} [44]	P_1, P_4
P_{24} [45]	P_5, P_{12}
P_{25} [46]	P_1, P_9, P_8, P_{12}
P_{26} [47]	P_3, P_9, P_{11}, P_{12}, P_{14}, P_{17}
P_{27} [48]	P_3, P_{14}, P_{26}
P_{28} [49]	P_{15}, P_{12}
P_{29} [50]	P_{18}, P_{15}, P_{12}

Table 2: Citations mapping within this systematic mapping

they were not really related to hyper-heuristics in Software Testing context. Therefore, only 22.65% [1] of articles were related to software testing, and only 3.54%[2] of the total articles were in the context of hyper-heuristics for Software Testing. Thus, the amount of articles returned and which are related to the use of hyper-heuristics in the context of Software Testing is very small, indicating

[1]Not considering the 3 articles obtained in snowballing process.

[2]Not considering the 3 articles obtained in snowballing process.

a clear need for more efforts in this regard.

article	title	reference
P_1	An experimental study of hyper-heuristic selection and acceptance mechanism for combinatorial t-way test suite generation	[16]
P_2	A multi-objective and evolutionary hyper-heuristic applied to the Integration and Test Order Problem	[17]
P_3	Deriving products for variability test of Feature Models with a hyper-heuristic approach	[51]
P_4	A Tabu Search hyper-heuristic strategy for t-way test suite generation	[52]
P_5	An orchestrated survey of methodologies for automated software test case generation	[53]
P_6	A survey on Test Suite Reduction frameworks and tools	[54]
P_7	The Evolutionary Landscape of SBST: a 10 Year Perspective	[15]
P_8	Hyperheuristics Search for SBST	[35]
P_9	A Hyper-Heuristic for the Multi-Objective Integration and Test Order Problem	[36]
P_{10}	Automatic Generation of Search-Based Algorithms Applied to the Feature Testing of Software Product Lines	[37]
P_{11}	Grammatical Evolution for the Multi-Objective Integration Test Order Problem	[38]
P_{12}	Learning Combinatorial Interaction Test Generation Strategies using Hyperheuristic Search	[55]
P_{13}	A Multi-objective optimization approach for selection of second order mutant generation strategies	[39]
P_{14}	Hyper-Heuristic Based Produt Selection for Software Product Line Testing	[40]
P_{15}	A Hyper-Heuristic for Multi-Objective Integration and Test Order Problem in Google Guava	[41]
P_{16}	Evaluating a Multi-Objective Hyper-Heuristic for the Integration and Test Order Problem	[42]
P_{17}	Product Selection Based on Upper Confidence Bound MOEA/D-DRA for Testing Software Product Lines	[56]
P_{18}	*Uma hiper-heurística de seleção de meta-heurísticas para estabelecer sequências de módulos para o teste de software*	[43]
P_{19}	A New Hybrid Algorithm for Software Fault Localization	[57]
P_{20}	Quality Improvement and Optimization of Test Cases: A Hybrid Genetic Algorithm Based Approach	[58]
P_{21}	An Empirical Analysis of the Mutation Operator for Run-time Adaptive Testing in Self-adaptive Systems	[59]
P_{22}	A Parameter Free Choice Function Based Hyper-Heuristic Strategy for Pairwise Test Generation	[44]
P_{23}	Automatically Generating Search Heuristics for Concolic Testing	[60]
P_{24}	Boosting Search Based Software Testing by Using Ensemble Methods	[45]
P_{25}	Concrete hyperheuristic framework for test case prioritization	[46]
P_{26}	Incorporating User Preferences in a Software Product Line Testing Hyper-Heuristic Approach	[47]
P_{27}	Multiple Objective Test Set Selection for Software Product Line Testing: Evaluating Different Preference-based Algorithms	[48]
P_{28}	A pattern-driven solution for designing multi-objective evolutionary algorithms	[49]
P_{29}	*Uma Solução Baseada em Hiper-Heurística para Determinar Ordens de Teste na Presença de Restrições de Modularização*	[50]

Table 3: Selected studies

Out of the 29 studies selected, 4 were classified as secondary studies. More-

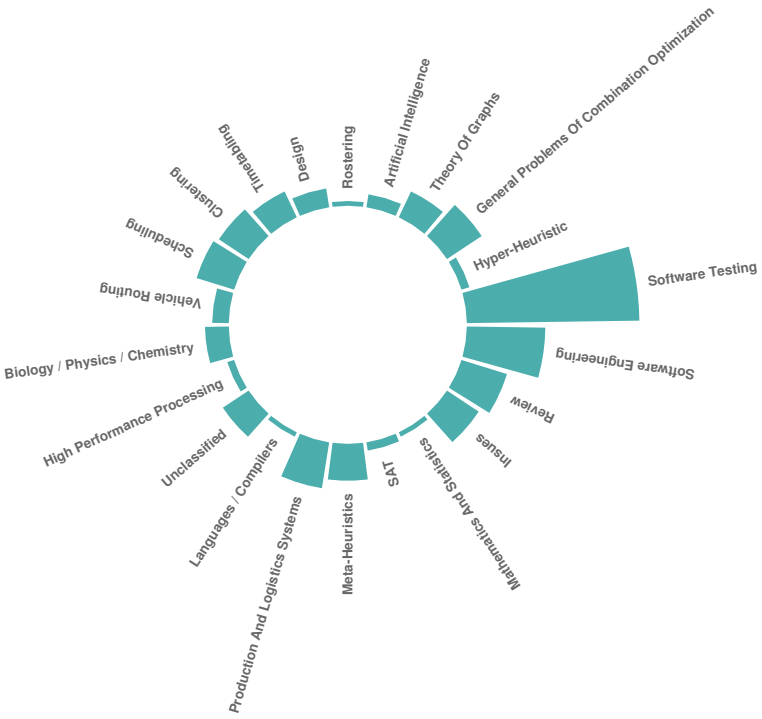

Figure 2: Distribution of articles: classes

over, 19 articles proposed/adapted hyper-heuristic approaches to solving problems such as ITO problem, CIT data generation, derivation of products for *Software Product Line* (SPL) testing, *Second Order Mutants* (SOM) generation strategies, Debbuging, Concolic Testing, Regression Testing, and Random Testing. The other studies are experimentations of the proposed hyper-heuristics.

Another general remark is that 56 authors from 28 different institutions were identified. Figure 3 presents the institutions of the authors of the articles published. There is a concentration of works published by a particular research group (Federal University of Paraná, Brazil) evidencing that it is necessary a greater spread of the efforts related to the use of hyper-heuristics within SBST, including the involvement of researchers from other communities such as

Operational Research and Artificial Intelligence.

Regarding the publication vehicles, the *Applied Software Computing*, *IEEE Congress on Evolutionary Computation* and the *International Workshop on Search-Based Software Testing* with published 3 papers, and this is the highest number of papers in a single vehicle. Other publication vehicles are *Journal of Systems and Software* (1), *Brazilian Conference on Intelligent Systems* (1), and *Brazilian Symposium on Software Engineering* (1). In addition, most of the studies were published in conferences (9), while the remaining are in journals (6) and workshops (2). This indicates that there is no specialized publication vehicle yet on this topic.

In the following sections, the research questions proposed in this systematic mapping are answered.

4.1. What are the hyper-heuristics used in the context of SBST? (RQ1)

To answer this question, it is important to adopt a classification for hyper-heuristics according to the main relevant characteristics of its implementation. The classification adopted in this work is in accordance with property $Prop_1$.

In addition to establishing a classification, the first step is to identify which studies propose or not a new hyper-heuristic. Out of the 29 studies, 19 propose/adapted a new hyper-heuristic approach (see Table 3 for the identification of the papers): P_1, P_3, P_4, P_9, P_{11}, P_{12}, P_{13}, P_{14}, P_{17}, P_{18}, P_{19}, P_{20}, P_{21}, P_{22}, P_{23}, P_{24}, P_{25}, P_{26} and P_{27}. The studies P_5-P_8 were classified as secondary studies. The other studies consist of the application of existing hyper-heuristics.

Based on the 19 studies that propose/adapted new hyper-heuristics, Table 4 was constructed where the properties of interest for the classification of each of the proposed hyper-heuristics are presented. The symbol "?" represents the information that could not be extracted from the papers.

It is important to stress that some approaches are not really defined as hyper-heuristics by the authors. However, some of them were selected to extract data aiming at increasing the number of selected papers. Characteristics such as hybridization and self-adaptiveness helped classifying them as hyper-heuristics.

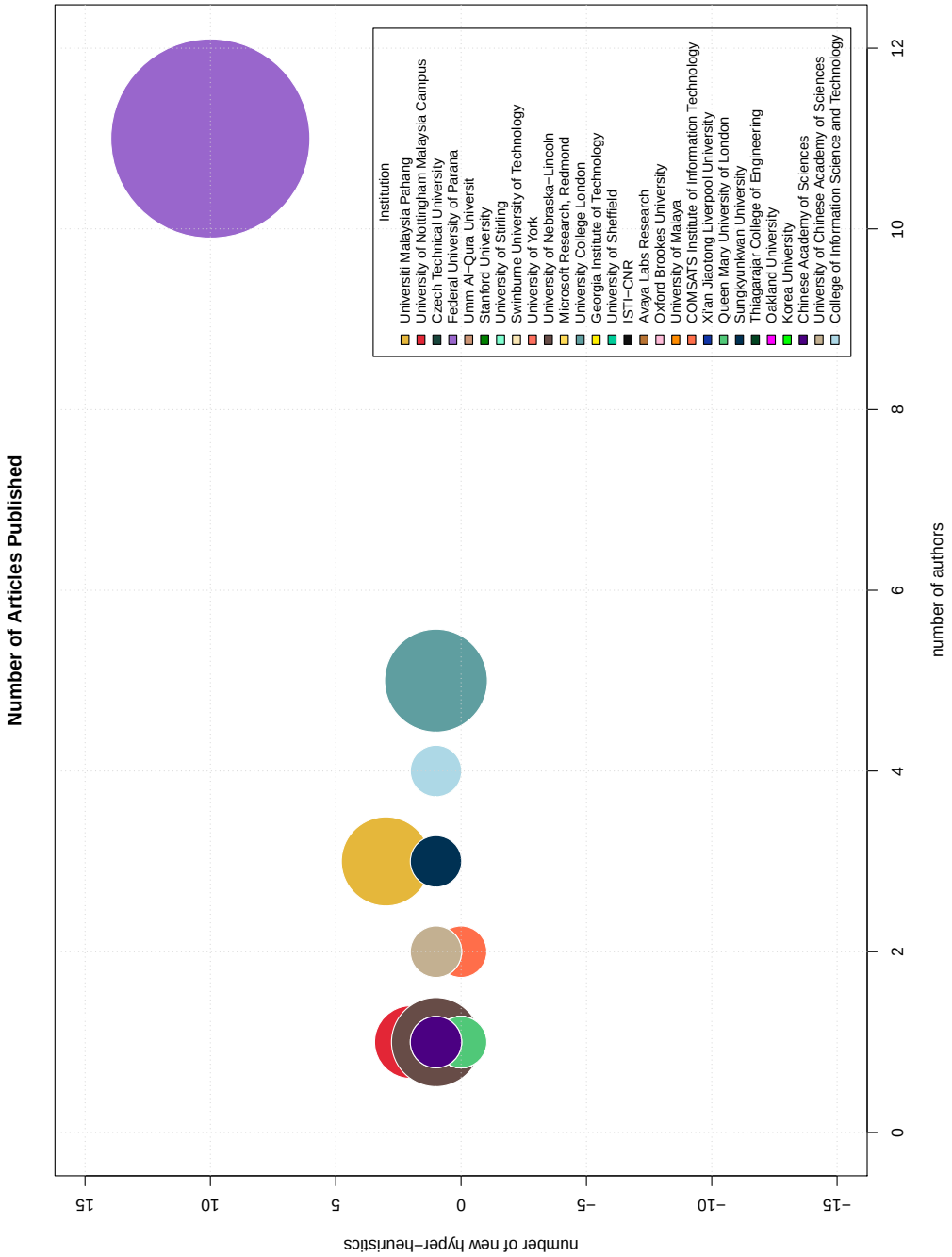

Figure 3: Distribution of published articles

article	type	learning	base	LLH	selection method
P_1	selection	online	?	Genetic Algorithm Crossover , Teaching Learning, Flower Global, Jaya	Exponential Monte Carlo, Choice Function, Improvement Selection Rules or Fuzzy Inference Selection
P_2	selection	online	NSGA-II	combinations of crossover and mutation operators	Choice Function or Multi-Armed Bandit
P_3	selection	online	NSGA-II	combinations of crossover and mutation operators	Random Select or FRR-Multi-Armed Bandit
P_4	selection	online	Tabu Search	Teaching Learning, Global Neighboohood, PSO, Cuckoo Search	Based on improvement, diversification and intensification
P_9	selection	online	NSGA-II	combinations of crossover and mutation operators	Random Select or Multi-Armed Bandit
P_{10}	generating	offline	Grammatical Evolution	combinations of crossover and mutation operators	-
P_{11}	generating	offline	Grammatical Evolution	combinations of crossover and mutation operators	-
P_{12}	selection	online	Simulated Anneling	combinations of crossover and mutation operators	Reinforcement Learning Agent
P_{13}	selection	online	NSGA-II	combinations of crossover and mutation operators	Choice Function
P_{14}	selection	online	NSGA-II, SPEA2, IBEA, and MOEA/D-DRA	combinations of crossover and mutation operators	Random Select or Upper-Confidence Bound
P_{15}	selection	online	NSGA-II	combinations of crossover and mutation operators	Random Select, Multi-Armed Bandit or Choice Function
P_{16}	selection	online	NSGA-II and SPEA2	combinations of crossover and mutation operators	Choice Function and Multi-Armed Bandit
P_{17}	selection	online	MOEA/D-DRA	combinations of crossover and mutation operators	Upper-Confidence Bound, Upper-Confidence Bound-Tuned or Upper-Confidence Bound-V
P_{18}	selection	online	MOCAITO	NSGA-II, SPEA-2 e IBEA	Choice Function
P_{19}	selection	online	?	Tarantula, AMPLE, Ochiai and Jaccard, among others 30	K-Means
P_{20}	selection	online	?	Simple Genetic Algorithm, Bacteriological Algorithm and local search	?
P_{21}	selection	online	Evolutionary Algorithm	combinations of crossover and mutation operators	?
P_{22}	selection	online	Choice Function	Genetic Algorithm Crossover, TLBO Learning, FPA global pollination, Jaya	Choice Function
P_{23}	generating	offline	Choice Function	A family of search heuristics	-
P_{24}	selection	online	?	Genetic Algorithm and SA	Performance Evaluation Chart
P_{25}	selection	online	Hierarquical Distribution	NSGA-ii, TAEA and SPEA2	?
P_{26}	selection	online	r-NSGA-II and PEMOA	combinations of crossover and mutation operators	?
P_{27}	selection	online	r-NSGA-II and PEMOA	combinations of crossover and mutation operators	?
P_{28}	generating	offline	Grammatical Evolution	combinations of crossover and mutation operators	-
P_{29}	selection	online	NSGA-II	combinations of crossover and mutation operators	Choice Function

Table 4: A classification of hyper-heuristics

With respect to $Prop_1$ (see Section 3.2), it is possible to note that the most used base algorithm was the *Non-dominated Sorting Genetic Algorithm-II*(NSGA-II) [61]. The NSGA-II algorithm was chosen as the base algorithm of the hyper-heuristics proposed in P_3 and P_{14} based on a comparative evaluation among other MOEAs that best solve instances of the target problem found in the literature. After this evaluation, in which the NSGA-II algorithm was the most successful, the basic algorithm was incorporated into the hyper-heuristics. In P_9, authors do not discuss the reasons why NSGA-II was used as the base algorithm. In P_{13}, authors justify the use of NSGA-II because it is the most commonly used MOEA in the context of SBSE [62].

Regarding the type of hyper-heuristic, and based on Table 4, only 4 studies, P_{10}, P_{11}, P_{23} and P_{28}, are not selection hyper-heuristics but the studies P_{10} and P_{28} are based on P_{11}. Thus, it can be realized that selection hyper-heuristics are often more used than generation hyper-heuristics. One of the reasons for this may be the fact that the supervised training, usually used in generation heuristics, may demand a lot of time, a fact that can contribute with the greater preference of the selection approach.

With respect to other characteristic of $Prop_1$, LLH set, although by definition they may be composed of complete heuristics/meta-heuristics, 13 out of the 19 studies in which a new hyper-heuristic is proposed considered as LLH simple search operators derived from already existing heuristics: the most used ones have been mutation and crossover operators. They vary not only in type (such as two point crossover, swap mutation, etc.) as in the values they assume.

Only 6 papers considered a complete MOEA as LLH, the papers P_4, P_{18}, P_{19}, P_{20}, P_{24} and P_{25}. There is not much information as to why these operators are used, perhaps because there is still little use of hyper-heuristic approaches in the context of SBST and, therefore, there is no much information to justify a choice which makes it empirical.

Still on $Prop_1$, it can observed a certain relationship between the type and the learning of hyper-heuristics approaches: selection hyper-heuristics have been associated only to *online* learning. The study [38] not only confirms but also

justifies the above relationship, when discussing the fact that generation hyper-heuristics generally perform well in unknown instances of the problem, so some type of training is usually employed and therefore it relates to *offline* learning.

Although *online* is the widely used learning approach, it offers some disadvantages, such as the need to allocate resources to the training mechanism during problem solving, which is overcome with the use of *offline* learning which solves this by using a previously trained heuristic [38].

The approaches that make use of selection hyper-heuristics rely on selection mechanisms which form the selection hyper-heuristic. The most used selection mechanisms are three: *Choice Function* [16, 17, 39] which adaptively ranks the LLHs with measures which make a balance between exploitation and exploration; *Multi-Armed Bandit* (MAB) [51, 36, 41] which takes into account the average of the performance of the LLH evaluated; and *Random Select* [51, 40, 41].

The most used selection hyper-heuristic was proposed in P_9, *Hyper-heuristic for the Integration and Test Order Problem* (HITO). This approach was tried in three other studies, where it was used in the context of the Integration Testing. However, such an approach had no other application than in the context of integration testing. This proves the non-exploration of a positive characteristic of hyper-heuristics, i.e. generalization.

4.2. What are the problems addressed by such strategies? (RQ2)

The objective of this question is to evaluate what are the problems, in the context of software testing, that have been solved through hyper-heuristic approaches.

As previously mentioned, out of the 29 articles selected, 4 were secondary studies. Addressed problems are ITO problem (8), CIT data generation (4), derivation of products for SPL testing (6), SOMs generation strategies (1), Random Testing (1), Debugging (1), Concolic Testing (1), and Regression Testing (3). Figure 4 shows the distribution of problem classes solved in relation to the type of approach applied.

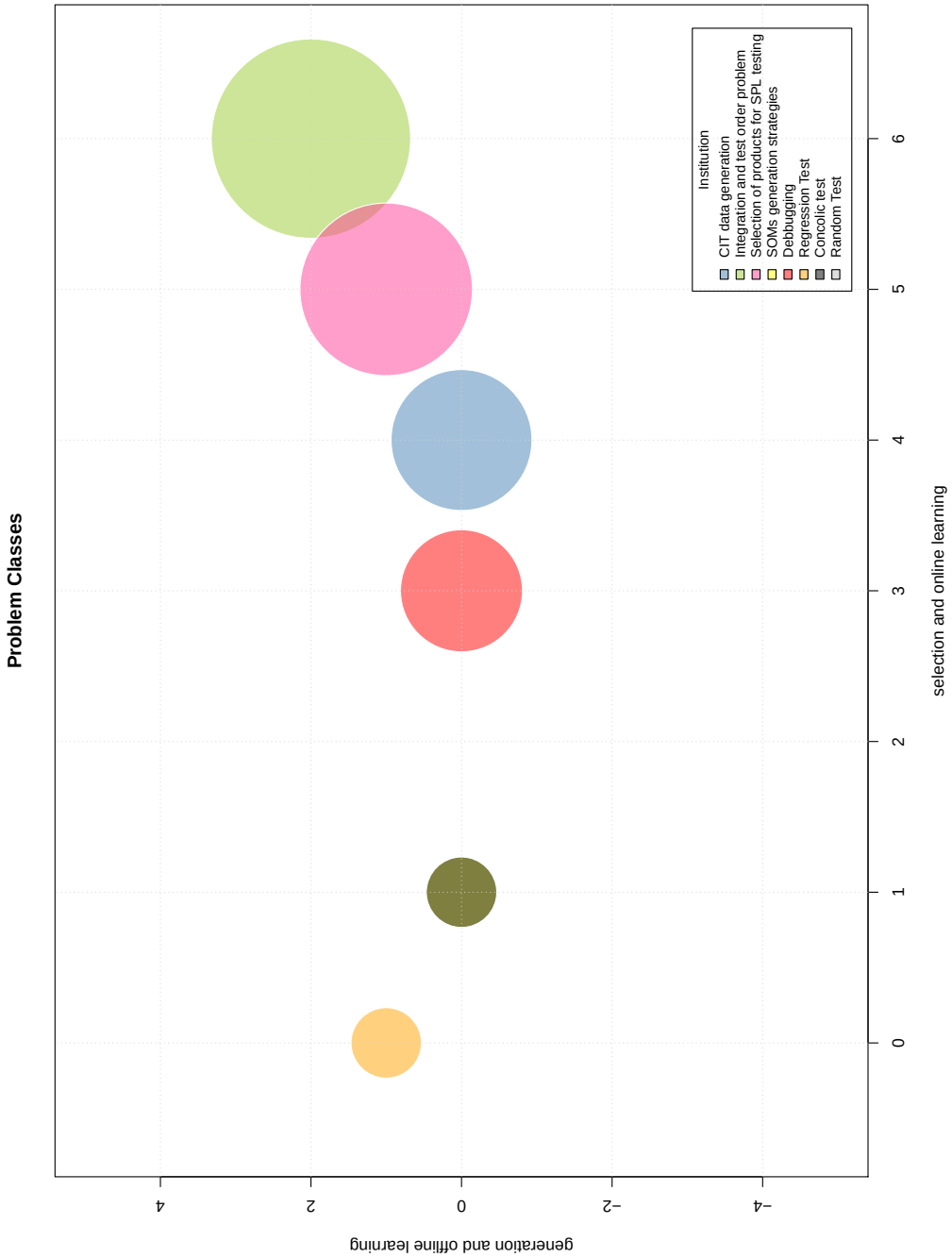

Figure 4: Problem classes

The problems that have been addressed by testing based on hyper-heuristics are described below:

(a) **ITO problem:** Integration Testing is one of the levels of Software Testing and its purpose is to verify that the smallest units of the software function properly when they are integrated. However, the order in which these units[3] are executed can influence in a series of factors, such as the masking of defects and the costs related to the execution time [17];

(b) **CIT data generation:** the motivation for CIT data generation is based on the idea that most of the faults can be revealed from the interaction of the parameters of the system considered. Thus, all parameter interactions and values of these parameters according to a certain degree of interaction must be present at least once in a *covering array*. [16];

(c) **Derivation of products for SPL testing:** SPL is commonly defined as a set of similar software (whose purpose is the same) constructed from the same base, which is defined by a set of requirements in common to them. The *feature models* Variability Test is a problem in which a set of possible products derived from an SPL is generated based on its requirements [37];

(d) **SOMs generation strategies:** the problem of *Higher-order Mutants* (HOMs) is to generate mutants from the combination of failures of other *First-order Mutants* (FOMs) (which generates n-*order Mutants*). The advantage of using this technique is to generate stronger mutants than the original ones, in addition to reducing the amount of mutants used [39];

(e) **Debugging:** Debugging is part of fault localization, a task of the testing activity [57]. Debugging consists of: (i) identification of the exact fault

[3]The authors consider the software developed under the Object-oriented Programming Paradigm, so the smallest parts of a software here are the *classes*.

location; and (ii) fixing the program defect and testing to confirm that the fault has been removed;

(f) **Random Testing:** Random Testing is to generate random, variable-based test cases. It is a Black Box testing strategy which has been receiving lots of attention from the community [63];

(g) **Regression Testing:** Regression Testing is one of the phases of the software testing activity that ensures that new defects have not been introduced during software maintenance [64]. Usually, Regression Testing is supported by the prioritization of test cases, in which certain test cases acquire execution priority;

(h) **Concolic Testing:** Concolic Testing is a test case generation technique that employs the symbolic execution of a program paired with its actual execution [46]. Its goal is to generate actual test cases based on code coverage.

Note that the range of problems is still very limited given the wide spectrum of software testing activities and tasks. For instance, no work has been published addressing the automated oracle problem. This is a problem which the testing community has long been addressing using other approaches. Mutation Testing is basically absent where only 1 paper addressed SOMs generation strategy. Hence, much more effort is required to properly perceive the advantages of hyper-heuristics for Software Testing.

Another important point is the application of a hyper-heuristic to a different class of problem from which it was proposed which, in theory, is to make use of one of the advantages of hyper-heuristics compared with the use of common heuristic algorithms: generalization. Only 1 study, P_{10}, explored this capability in the context of the selection of products for SPL testing, by using the hyper-heuristic originally proposed in P_{11} for the solution of the ITO problem. Thus, there is still little exploration of the advantages of using hyper-heuristics in this context.

4.3. What are the objective functions considered? (RQ3)

The purpose of this question is to evaluate the main test objectives in the context of the use of hyper-heuristics within SBST. At this point it is important to take into account the type of problem to be solved. Table 5 organizes the issues addressed and their respective test objectives. In summary, the objective functions the papers have addressed so far are: (i) CIT problem: number of tuples not yet covered; (ii) ITO problem: number of atributes, number of methods, number of return types, and number of parameter types; (iii) selection of products for SPL testing: number of products, pairwise coverage, mutation score, and product dissimilarity; (iv) SOMs generation strategies: SOM number, effectiveness, replacement capacity; (v) Regression Test: mutation score, statement coverage, computational cost of test case, past fault detection history, and path coverage; (vi) Concolic Testing: branch coverage, and bug-finding, and; (vii) Random Testing: affiliate coverage criterion.

Although 19 new hyper-heuristics have been proposed/adapted, other studies among the 29 articles selected in this systematic mapping make use of some of these proposed hyper-heuristics, and the great majority uses equal objectives to solve the same problem class. However, with respect to the problem of selection of products for SPL testing, which are addressed by P_3 and P_{14}, there is a peculiarity: P_{14} takes into account an additional goal in relation to P_3, as shown in Table 5. Moreover, it is highlighted in P_{14} that unlike the other studies, the authors consider product dissimilarity as a test goal, since testing one set of dissimilar products from one another may offer a greater probability of finding non-standard errors disclosed in SPL.

Out of the 8 classes of problems found, only the CIT class is modeled as a single objective problem. The other problems are multi-objective. This is a good strategy because a multi-objective scenario is a more realistic one. One can also note this by taking into account the basic algorithms used in each of the hyper-heuristic approaches associated with the CIT problem: P_4 uses Tabu Search and P_{12} uses Simulated Anneling, both algorithms were used to solve single objective problems (Table 4). The other approaches that solve the remaining

article	case study	comparation	objective function	class problem
P_1	Case studies used in a previous study	MH ×HH	number of tuples not yet covered	CIT data generation
P_2	MyBatis, AJHSQLDB, AJHotDraw, BCEL, JHotDraw, HealthWatcher and Jboss	MH ×HH	number of attributes, number of methods, number of return types, and number of parameter types to be emulated	ITO problem
P_3	James, CAS, WS e E-Shop	MH ×HH	number of products, pairwise coverage, mutation score	SPL testing
P_4	Case studies used in a previous study	MH ×HH	number of tuples not yet covered	CIT data generation
P_9	MyBatis, AJHSQLDB, AJHotDraw, BCEL, JHotDraw, HealthWatcher and Jboss	MH ×HH	number of atributes and number of methods	ITO problem
P_{10}	James, CAS, WS e E-Shop	MH ×HH	number of products, pairwise coverage, mutation score and number	SPL testing
P_{11}	MyBatis, AJHSQLDB, AJHotDraw, BCEL, JHotDraw, HealthWatcher and Jboss	MH ×HH and HH ×HH	number of atributes, number of methods	ITO problem
P_{12}	used 5 sets of test cases with and without the use of constraints	MH ×HH	number of tuples not yet covered	CIT data generation
P_{13}	Bisect, bub, find, fourballs, mid and triangle	MH ×HH	SOM number, effectiveness, replacement capacity	SOMs generation strategies
P_{14}	James, CAS, WS e E-Shop	MH ×HH	number of products, pairwise coverage, mutation score, product dissimilarity	SPL testing
P_{15}	Google Guava	MH ×HH	number of atributes, number of methods	ITO problem
P_{16}	MyBatis, AJHSQLDB, AJHotDraw, BCEL, JHotDraw, HealthWatcher and Jboss	MH ×HH	number of atributes, number of methods	ITO problem
P_{17}	James, CAS, WS e E-Shop	MH ×HH	number of products, pairwise coverage, mutation score	SPL testing
P_{18}	AJHotDraw, AJHSQLDB, BCEL and MyBatis	MH ×HH	number of atributes, number of methods, number of return types, number of parameter types	ITO problem
P_{19}	Siemens test suite	MH ×HH	?	Debbugging
P_{20}	5 case studies of the real world in Java	MH ×HH	?	Regression Test
P_{21}	Remote Data Mirroring and Smart Vacuum System	results validation	?	Regression Test
P_{22}	benchmarks used in various studies	MH ×HH	describes only the metrics for the selection of the search operators	CIT data generation
P_{23}	10 previously used benchmarks	MH ×HH	branch coverage and bug-finding.	Concolic Test
P_{24}	?	MH ×HH	level+branch distance, dissimilarity distance, symbolic enhance distance	Random Test
P_{25}	8 case studies (5 in java and 3 in c ++) - real industrial projects	MH ×HH	APSC (scalability) e EET (mensura o custo da execuo)	Regression Test
P_{26}	James, CAS, WS e E-Shop	MH ×HH	number of products, mutation score and pairwise coverage	SPL testing
P_{27}	James, CAS, WS e E-Shop	MH ×HH	number of products, mutation score, pairwise coverage and similarity	SPL testing
P_{28}	7 real-world case studies written in Java and AspectJ	?	was applied to two different case studies	ITO problem
P_{29}	MyBatis, AJHSQLDB, AJHotDraw, BCEL, JHotDraw, HealthWatcher and Jboss	MH ×HH	number of atributes and number of methods	ITO problem

Table 5: Context of application of hyper-heuristics

problems have, as basis algorithm, multi-objective ones such as NSGA-II.

This outcome is also interesting because even in the wider engineering and design problems in general, hyper-heuristics have been used to deal with single objective problems, with a few attempts to apply hyper-heuristics to multi-objective problems [65, 66].

Thus, despite the relatively small amount of work in the context of software testing, in 78.94% of these, the problems are multi-objective. This is indicative that the SBST community is trying to solve the testing problems in a more comprehensive way.

4.4. Which artifacts (source code, models, ...) they relied on? (RQ4)

The purpose of this question is to evaluate which are the main software artifacts used as input to the hyper-heuristic approaches. These artifacts are associated with the type of problem solved (see previous section).

The software artifacts considered in the ITO problem are usually graphs that illustrate the relationship between each of the units of the system considered, known as *Object Relation Diagram* (ORD). The vertices represent the units and the edges represent the relationships between these units [36].

The problem of CIT data generation takes into account only the quantity of parameters, values of these parameters, and degree of interaction (strength). With respect to the derivation of products for SPL testing, feature models, which are tree structures that hierarchize relevant characteristics of a given product (in this case, software), are considered.

In accordance with [39], the artifacts which are taken into account for the SOMs generation strategies problem are the source code, the test cases, the FOM set, and the metadata.

Finally, Debugging takes into account only the code, while Regression Testing takes into account the test suite and the source code, and random test case generation takes into account only the quantity of parameters and values of these parameters.

4.5. What is the validation context related to the application of hyper-heuristics for Software Testing? (RQ5)

Table 5 presents the answer for this question. Note that the only hyper-heuristics that had more than one study published were HITO [36] and *Grammatical Evolution for the Multi-Objective Integration and Test Order Problem* (GEMOITO) [38]. Studies P_2, P_{15}, P_{16} and P_{29} use the HITO hyper-heuristic proposed in P_9. Thus, one of the differences between each of these studies is due to the validation context. Initially, in P_9 it was used 7 real case studies, where the number of units and dependencies of each was taken into consideration for their selection. In P_9, there are 2 research questions regarding the performance of the HITO and which is the best selection/acceptance mechanism used. In P_2, the case studies are exactly the same as those used in P_9, however, the study considers a third research question, intended to evaluate the set of more appropriate LLHs. P_{15} takes into account the Google Guava software, a set of libraries for Java. In P_{16} the same 7 case studies used in P_9 are used, however, the purpose here is to evaluate the performance of the HITO hyper-heuristic using as base algorithm different MOEAs, in contrast to the original study comparing HITO from the point of view of different selection/acceptance mechanisms, and to P_2 that compares HITO under different sets of LLHs. With respect to the studies derived from GEMOITO [38], their differences lie in the adaptation of the problem, as already mentioned.

Based on the low number of case studies used for the validation of the proposed hyper-heuristics, it is possible to state that the hyper-heuristic validation context is restricted too. Studies related to the HITO hyper-heuristic provide the basis for this claim, since P_2 and P_{16} make use of the same case studies as P_9, however considering different topics. Only P_{15} applied to a different validation context.

The only study comparing the results of his own approach with another hyper-heuristic was P_{11}. This is another indication that the use of hyper-heuristics in the SBST community is limited, and thus there is not a sufficiently large variability of hyper-heuristics aimed at solving the same class of problems

in a way that a comparison is possible between approaches.

5. Future directions to SBST based on hyper-heuristics

In this section, some future research directions to the SBST community based on hyper-heuristics are present. Note that these suggestions may also be tailored to SBSE problems in general.

5.1. Development and use of generation hyper-heuristics for Software Testing

Based on the results of this systematic mapping study, it was possible to verify that the number of proposed generation hyper-heuristics and their applications within the Software Testing activity have been low in relation to the use of selection hyper-heuristics. It may be an important point to systematically check the reasons behind this difference in this preference, since some authors argue that generation hyper-heuristics possess more features for greater level of generalization compared with selection hyper-heuristics [10]. A possible explanation for the greater number of selection compared with generation strategies is that, usually, selection hyper-heuristics are easier to implement compared with the generation ones. But, it is necessary to certify that this is the only reason for choosing between these two options.

Moreover, it is not mandatory to create new generation hyper-heuristics but rather to adapt solutions already used in other problems (production scheduling, cutting and packing, ...) to Software Testing and realize about their effectiveness, efficiency, and cost. For instance, Genetic Programming [67, 68] has been used as a generation hyper-heuristics in some of these problems. Is it useful for software test case/data generation?

5.2. Using complete MOEAs as LLHs

Only 6 studies considered an entire MOEA as LLH in selection hyper-heuristics for testing. This is another approach that can be further investigated because the benefits of the entire MOEA can be take into account. Note

that in other problems, such as assessing the performance considering a common benchmark for multi-objective optimization and vehicle crashworthiness design problem [69], and wind farm layout optimization [70], selection hyper-heuristics with complete MOEAs (e.g. NSGA-II, *Strength Pareto Evolutionary Algorithm*(SPEA2)) as LLHs have obtained better performance in most of the cases when comparing them with each constituent MOEA used on its own (i.e. in isolation).

Moreover, for selection hyper-heuristics, it is necessary to create an adequate mechanism to transfer the populations from one MOEA into another one, in cases where the selected MOEA to run is different from the last MOEA executed. While some MOEAs deal with a population, others, in addition to the population, make use of an archive which has a representation of the non-dominated front among all solutions considered so far [71]. Hence, it is required to devise a solution to properly handle this situation. In [43], the authors proposed such a mechanism, but it is not totally clear how effective is their proposal.

Besides the few approaches that relied on generation hyper-heuristics, the heuristics the techniques generate are either components of an Evolutionary Algorithm (such as crossover and mutation operators) [37, 38] or search heuristics for Concolic Testing [46]. Thus, generating entire MOEAs rather than only components of it is worth to be investigated in the generation context as well.

5.3. Investigating the generalization characteristic of hyper-heuristics

As previously pointed out, one of the appealing reasons for choosing hyper-heuristics over meta-heuristics is the capability of generalization of hyper-heuristics. According to the outcomes of this secondary study, only 1 study did it in the context of the selection of products for SPL testing, by using a hyper-heuristic originally proposed for the solution of the ITO problem.

Thus, it is clearly necessary to evaluate the true potential of generalization of hyper-heuristics strategies. Hence, for instance, a selection hyper-heuristic used for test case/data generation addressing *Graphical User Interface* (GUI) appli-

cations is adequate and presents good performance, with no additional tuning of parameters, to work as an automated oracle for embedded software systems? Or, a generation hyper-heuristic used for test case prioritization of scientific software is good, with no additional tuning of parameters, for Robustness Testing in cloud platforms? These are some examples that should be addressed in the future by the academic community.

Opponents of hyper-heuristics may argue that there is still a need for parameterization when trying to apply a certain hyper-heuristic to different problems. Consider Choice Function, one of the most widely used methods of selection in hyper-heuristics in the context of Software Testing. Although in its initial proposal the Choice Function parameters (α, β related to intensification/exploitation; δ related to diversification/exploration) are dynamically adapted based on a reinforcement learning approach [72], in some works applied to software testing (e.g. for the ITO problem [17, 43]), there is a simplified version of the Choice Function method, where the parameters are chosen manually. So, there is still a need for tuning parameters but the level of abstraction is higher (in the Choice Function method) rather than tuning in a lower level (crossover and mutation operators and respective probabilities within a MOEA in isolation).

This fact is corroborated by recently proposed selection hyper-heuristics, applied to optimization benchmarks and vehicle crashworthiness problems, which also require tuning of parameters [73]. This is a point that is relevant to be investigated by the Software Testing community, since the conclusions in this sense can be of great benefit to other communities involved in the optimization area.

5.4. Evaluating effectiveness of test suites

To compare the performance of the proposed hyper-heuristics with other strategies, quality indicators such as *hypervolume*, *Inverted Generational Distance* (IGD), *spread* are traditionally measured by the optimization researchers. Software testers have also done it in accordance with the papers returned by

this systematic mapping.

However, the Software Testing community is used to rely on metrics such as effectiveness (e.g. ability to identify faults in the source code, code coverage) to compare if a test suite due to an approach A is better than other test suite due to another strategy B. No study selected for data extraction in this systematic mapping have performed this appropriately as presented below.

While some reports which aim at deriving products for SPL testing [51, 40, 56] had as one objective the minimization of the number of alive mutants, and in [39] one objective function is the capacity of the SOM to reveal subtler faults, i.e. faults not revealed by its constituent FOMs and hence is an effectiveness measure, it is not totally clear what is the effectiveness of selection hyper-heuristics in other problems or validation contexts compared with meta-heuristic approaches or even other strategies.

The minimization of the number of alive mutants in the SPL testing problem as mentioned above is an effectiveness metric but the problem of this objective function is the cost to obtain it, since it is required to know the number of killed mutants and the total number of generated non-equivalent mutants. Moreover, it is very likely that a lot preprocessing steps must have done in order to format the data suitable to be treated by the hyper-heuristic algorithm.

Hence, one interesting direction is to consider less costly effectiveness metrics as objective functions such as test case diversity aided by similarity measures [74]. In this case, at first, there is no need to demand preprocessing efforts since one wants to minimize the pairwise similarity between test cases (i.e. solutions) of a test suite (population, sets of solutions). Thus, this can be done with no previous processing tasks.

5.5. Increasing the number of problems and validation contexts

As mentioned in Section 4.2, the range of problems is very limited given the wide spectrum of software testing activities and tasks. The automated oracle problem is not addressed at all up to now. This is a very important task of the testing activity which has long been addressed where specification-

based, metamorphic relation-based, *Machine Learning*-based types of oracles have been used [75]. However, no selection or generation hyper-heuristic is helping in this regard. A full automated testing activity needs that not only the test input data are generated but also to automatically identify the expected results (oracle information) and the ability to assert that test cases pass or not (oracle procedure).

Even though a mutation operator is one component of an Evolutionary Algorithm/Genetic Algorithm, hyper-heuristics have not been applied to the Mutation Testing where only 1 paper addressed SOMs generation strategy via a selection hyper-heuristic. Even in the context of HOMs/SOMs, an issue is the high cost of HOM testing. Determining whether a selection or generation hyper-heuristic is more effective for generating HOMs is an open issue. Again, meta-heuristics such as Genetic Algorithm [76], Simulated Anneling [77], and ACO [78] have been used for generating test input data for killing FOMs and HOMs but no hyper-heuristic approach did it. Detecting higher-order equivalent mutants is an open question within the testing community as a whole. Hence, this is another direction where hyper-heuristics may help.

Increasing the validation context is very important too. For instance, from the 8 papers on ITO problem, 5 considered the very same case studies. All papers on derivation of products for SPL testing took into account the same feature models. Hence, it is clear that it is necessary to further investigate more problems and validation contexts by using hyper-heuristics.

5.6. Addressing non-functional requirements/properties

Non-functional requirements of a system can be understood as its quality attributes, i.e. a set of concerns related to the concept of quality [79]. Testing a software system is an effort that should involve not only its functional characteristics but also its non-functional properties.

Within the SBST community, non-functional requirements testing have been done as presented by Afzal et al. (2009) [80] in their SLR. The authors identified five different categories of non-functional requirements where researchers

paid attention: execution time, quality of service, security, usability, and safety. Genetic Algorithm, Simulated Anneling, ACO, Tabu Search, and PSO are some meta-heuristics used aiming at testing the system under this respect.

Later, Harman et al. (2015) [1] extended the review presented by Afzal et al. (2009) and identified 6 more categories: availability, efficiency, energy consumption, flexibility, robustness, and scalabilty.

In both reviews, there is no mention of any hyper-heurisitic strategy for testing non-funcional requirements/properties. Eventually, a multi-objective problem solved by hyper-heuristics where some objective functions are related to functional requirements while others deal with non-functional properties is an interesting path to follow, since the solutions for such a problem may target both characteristics of software systems at once.

6. Conclusions

This article presented an systematic mapping whose purpose was to investigate the use of hyper-heuristics in the context of SBST. At first, 734 articles were returned from 4 different databases (ACM Digital Library, IEEEXplore, ScienceDirect, and Scopus), of which only 29 were within the context of this research. The research questions defined in this study seek to clarify which are the main hyper-heuristic approaches used, what are the problems addressed, what are the test objectives (objective functions), what artifacts are related to the application of hyper-heuristics, and what is the validation context.

In general, one can say that among the advantages of using hyper-heuristic approaches is the fact that they are more general than meta-heuristic solutions and can be applied to a class of problems rather than a single specific problem. However, this feature has been little explored in the context of SBST, since only one article makes use of the hyper-heuristic proposed in another one, developed to solve the ITO problem, however in the context of the SPL testing. This may be associated with the fact that the use of hyper-heuristics is still very premature, so some aspects have not yet been explored consistently by the

Software Testing researchers.

The main findings of this systematic mapping study are presented below:

- Selection hyper-heuristics with *online* learning has prevailed. This kind of approach allows successful common algorithms to solve a given problem by working together. In addition, this type of approach makes use of two mechanisms: (i) selection mechanism; and (ii) acceptance mechanism. Both can vary which provides a greater generality of the solutions produced. In addition, selection hyper-heuristics with *online* learning do not require training, which facilitates their usage;

- Only 6 studies were generation hyper-heuristics with *offline* learning. This type of configuration offers some advantages over the *selection* configuration with *online* learning: no need to choose existing algorithms, no configuration of the parameters of these algorithms, no allocation of resources for simultaneous training execution;

- The validation context related to hyper-heuristic is restricted. In most cases, previous case studies have been used in other future studies. In addition, the experiments did not take into account the hyper-heuristic $\times$ hyper-heuristic comparison[4];

- The addressed problems related to Software Testing is also restricted, and it is limited to 8 problems: ITO problem, CIT data generation, selection of products for SPL testing, SOMs generation strategies, Regression Testing, Random Testing, Concolic Testing, and Debugging. Oracle problem and other ones have no yet been addressed. Although hyper-heuristics have as main appeal a greater generality than common meta-heuristics, only one study made use of an already existing hyper-heuristic approach to solve a different problem;

- In the 19 articles where a new hyper-heuristic is proposed/adapted, the

[4]Comparison between approaches of the same type: hyper-heuristic.

LLH set was composed only of simple search operators, and only in 6 case the complete/entire meta-heuristic is used;

- The definition of hyper-heuristic as proposed by Burke et al. (2010) [11] and Burke et al. (2013) [10] was the main driving factor to select the papers for data extraction. But, even if some approaches are not really defined as hyper-heuristics by the researchers, they were selected aiming at increasing the number of papers. Characteristics such as hybridization and self-adaptiveness helped in this regard;

- It is possible to conclude, based on the low number of articles that rely on hyper-heuristics for helping the Software Testing activity, that the SBST researchers have not explored the benefits of this technique. Another factor that corroborates this fact is the publication date of the articles: the oldest study is from the year 2010. Therefore, the use of hyper-heuristics in the context of SBST is still taking its first steps.

In Section 5, suggestions of future directions to SBST based on hyper-heuristics are proposed. The authors believe that this further research is relevant to the area of testing, or in some cases even for SBSE, so that new methodologies/methods/techniques can be created and assessments can be made to really perceive the benefits of this optimization theory for Software Testing.

References

[1] M. Harman, Y. Jia, Y. Zhang, Achievements, open problems and challenges for search based software testing, in: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), 2015, pp. 1–12. doi:10.1109/ICST.2015.7102580.

[2] J. Miller, M. Reformat, H. Zhang, Automatic test data generation using genetic algorithm and program dependence graphs, Information and Software Technology 48 (7) (2006) 586 – 605. doi:https://doi.org/10.1016/j.infsof.2005.06.006.

[3] S. Yoo, M. Harman, Pareto efficient multi-objective test case selection, in: Proceedings of the 2007 International Symposium on Software Testing and Analysis, ISSTA '07, ACM, New York, NY, USA, 2007, pp. 140–150. doi:10.1145/1273463.1273483.

[4] A. Windisch, S. Wappler, J. Wegener, Applying particle swarm optimization to software testing, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO '07, ACM, New York, NY, USA, 2007, pp. 1121–1128. doi:10.1145/1276958.1277178.

[5] T. Mahmoud, B. S. Ahmed, An efficient strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software testing use, Expert Systems with Applications 42 (22) (2015) 8753 – 8765. doi:https://doi.org/10.1016/j.eswa.2015.07.029.

[6] M. Patil, P. Nikumbh, Pair-wise testing using simulated annealing, Procedia Technology 4 (2012) 778 – 782, 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012. doi:https://doi.org/10.1016/j.protcy.2012.05.127.

[7] J. Petke, M. B. Cohen, M. Harman, S. Yoo, Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection, IEEE Transactions on Software Engineering 41 (9) (2015) 901–924. doi:10.1109/TSE.2015.2421279.

[8] E. Daz, J. Tuya, R. Blanco, J. J. Dolado, A tabu search algorithm for structural software testing, Computers & Operations Research 35 (10) (2008) 3052 – 3072, part Special Issue: Search-based Software Engineering. doi:https://doi.org/10.1016/j.cor.2007.01.009.

[9] L. G. Hernandez, N. R. Valdez, J. T. Jimenez, Construction of mixed covering arrays of variable strength using a tabu search approach (2010).

[10] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: a survey of the state of the art, Journal of the Operational Research Society 64 (12) (2013) 1695–1724. doi:10.1057/jors.2013.71.

[11] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, J. R. Woodward, A classification of hyper-heuristic approaches (2010). doi:10.1007/978-1-4419-1665-5_15.

[12] A. Garca-Villoria, S. Salhi, A. Corominas, R. Pastor, Hyper-heuristic approaches for the response time variability problem, European Journal of Operational Research 211 (1) (2011) 160 – 169. doi:https://doi.org/10.1016/j.ejor.2010.12.005.

[13] P. Garrido, M. C. Riff, Dvrp: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic, Journal of Heuristics 16 (6) (2010) 795–834. doi:10.1007/s10732-010-9126-2.

[14] R. Tyasnurita, E. Ozcan, A. Shahriar, R. John, Improving performance of a hyper-heuristic using a multilayer perceptron for vehicle routing (09 2015).

[15] M. B. Cohen, The evolutionary landscape of sbst: A 10 year perspective, in: 2017 IEEE/ACM 10th International Workshop on Search-Based Software Testing (SBST), 2017, pp. 47–48. doi:10.1109/SBST.2017.2.

[16] K. Z. Zamli, F. Din, G. Kendall, B. S. Ahmed, An experimental study of hyper-heuristic selection and acceptance mechanism for combinatorial t-way test suite generation, Information Sciences 399 (2017) 121 – 153. doi:https://doi.org/10.1016/j.ins.2017.03.007.

[17] G. Guizzo, S. R. Vergilio, A. T. Pozo, G. M. Fritsche, A multi-objective and evolutionary hyper-heuristic applied to the integration and test order problem, Applied Soft Computing 56 (2017) 331 – 344. doi:https://doi.org/10.1016/j.asoc.2017.03.012.

[18] P. Gupta, I. Arora, A. Saha, A review of applications of search based software engineering techniques in last decade, in: 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 2016, pp. 584–589. doi:10.1109/ICRITO.2016.7785022.

[19] A. Saeed, S. H. A. Hamid, M. B. Mustafa, The experimental applications of search-based techniques for model-based testing: Taxonomy and systematic literature review, Applied Soft Computing 49 (2016) 1094 – 1117. doi:https://doi.org/10.1016/j.asoc.2016.08.030.

[20] V. Arora, R. Bhatia, M. Singh, A systematic review of approaches for testing concurrent programs, Concurr. Comput. : Pract. Exper. 28 (5) (2016) 1572–1611. doi:10.1002/cpe.3711.

[21] F. Popentiu-Vladicescu, G. Albeanu, Nature-inspired approaches in software faults identification and debugging, Procedia Computer Science 92 (2016) 6 – 12, 2nd International Conference on Intelligent Computing, Communication & Convergence, ICCC 2016, 24-25 January 2016, Bhubaneswar, Odisha, India. doi:https://doi.org/10.1016/j.procs.2016.07.315.

[22] R. A. Silva, S. do Rocio Senger de Souza, P. S. L. de Souza, A systematic review on search based mutation testing, Information and Software Technology 81 (2017) 19 – 35. doi:https://doi.org/10.1016/j.infsof.2016.01.017.

[23] P. Vats, M. Mandot, A. Gosain, A comparative analysis of ant colony optimization for its applications into software testing, in: 2014 Innovative Applications of Computational Intelligence on Power, Energy and Controls with their impact on Humanity (CIPECH), 2014, pp. 476–481. doi:10.1109/CIPECH.2014.7019110.

[24] M. Khari, P. Kumar, An extensive evaluation of search-based software testing: a review, Soft Computing 23 (6) (2019) 1933–1946. doi:10.1007/

s00500-017-2906-y.
URL https://doi.org/10.1007/s00500-017-2906-y

[25] M. Harman, B. Jones, Search based software engineering ?, Information and Software Technology 43 (14) (2001) 833 – 839.

[26] J. Ferrer, F. Chicano, E. Alba, Evolutionary algorithms for the multi-objective test data generation problem, Softw. Pract. Exper. 42 (11) (2012) 1331–1362. doi:10.1002/spe.1135.

[27] N. Asoudeh, Y. Labiche, Multi-objective construction of an entire adequate test suite for an efsm, in: 2014 IEEE 25th International Symposium on Software Reliability Engineering, 2014, pp. 288–299. doi:10.1109/ISSRE.2014.14.

[28] A. Shahbazi, J. Miller, Black-box string test case generation through a multi-objective optimization, IEEE Transactions on Software Engineering 42 (4) (2016) 361–378. doi:10.1109/TSE.2015.2487958.

[29] B. S. Ahmed, L. M. Gambardella, W. Afzal, K. Z. Zamli, Handling constraints in combinatorial interaction testing in the presence of multi objective particle swarm and multithreading, Information and Software Technology 86 (2017) 20 – 36. doi:https://doi.org/10.1016/j.infsof.2017.02.004.

[30] P. McMinn, Search-based software testing: Past, present and future, in: Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, ICSTW '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 153–163. doi:10.1109/ICSTW.2011.100.

[31] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE'08,

BCS Learning & Development Ltd., Swindon, UK, 2008, pp. 68–77.
URL http://dl.acm.org/citation.cfm?id=2227115.2227123

[32] T. Mariani, S. R. Vergilio, A systematic review on search-based refactoring, Information and Software Technology 83 (2017) 14 – 34. doi:https://doi.org/10.1016/j.infsof.2016.11.009.

[33] L. E. G. Martins, T. Gorschek, Requirements engineering for safety-critical systems: A systematic literature review, Information and Software Technology 75 (2016) 71 – 89. doi:https://doi.org/10.1016/j.infsof.2016.04.002.

[34] C. B. Begg, The Handbook of Research Synthesis, H. Cooper & L. Hedges, USA, 1994.

[35] Y. Jia, Hyperheuristic search for sbst, in: Proceedings of the Eighth International Workshop on Search-Based Software Testing, SBST '15, IEEE Press, Piscataway, NJ, USA, 2015, pp. 15–16.

[36] G. Guizzo, G. M. Fritsche, S. R. Vergilio, A. T. R. Pozo, A hyper-heuristic for the multi-objective integration and test order problem, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO '15, ACM, New York, NY, USA, 2015, pp. 1343–1350. doi:10.1145/2739480.2754725.

[37] H. L. J. Filho, J. A. P. Lima, S. R. Vergilio, Automatic generation of search-based algorithms applied to the feature testing of software product lines, in: Proceedings of the 31st Brazilian Symposium on Software Engineering, SBES'17, ACM, New York, NY, USA, 2017, pp. 114–123. doi:10.1145/3131151.3131152.

[38] T. Mariani, G. Guizzo, S. R. Vergilio, A. T. Pozo, Grammatical evolution for the multi-objective integration and test order problem, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016,

GECCO '16, ACM, New York, NY, USA, 2016, pp. 1069–1076. doi:10.1145/2908812.2908816.

[39] J. A. P. Lima, S. R. Vergilio, A multi-objective optimization approach for selection of second order mutant generation strategies, in: Proceedings of the 2Nd Brazilian Symposium on Systematic and Automated Software Testing, SAST, ACM, New York, NY, USA, 2017, pp. 6:1–6:10. doi:10.1145/3128473.3128479.

[40] T. N. Ferreira, J. A. P. Lima, A. Strickler, J. N. Kuk, S. R. Vergilio, A. Pozo, Hyper-heuristic based product selection for software product line testing, IEEE Computational Intelligence Magazine 12 (2) (2017) 34–45. doi:10.1109/MCI.2017.2670461.

[41] G. Guizzo, M. Bazargani, M. Paixao, J. H. Drake, A hyper-heuristic for multi-objective integration and test ordering in google guava, in: T. Menzies, J. Petke (Eds.), Search Based Software Engineering, Springer International Publishing, Cham, 2017, pp. 168–174.

[42] G. Guizzo, S. R. Vergilio, A. T. R. Pozo, Evaluating a multi-objective hyper-heuristic for the integration and test order problem, in: 2015 Brazilian Conference on Intelligent Systems (BRACIS), 2015, pp. 1–6. doi:10.1109/BRACIS.2015.11.

[43] V. R. De Carvalho, S. Vergilio, A. Pozo, Uma hiper-heurstica de seleo de meta-heursticas para estabelecer sequncias de mdulos para o teste de software (09 2015).

[44] F. Din, A. R. A. Alsewari, K. Z. Zamli, A parameter free choice function based hyper-heuristic strategy for pairwise test generation, in: 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2017, pp. 85–91. doi:10.1109/QRS-C.2017.22.

[45] X. Xu, L. Jiao, Z. Zhu, Boosting search based software testing by using ensemble methods, 2018, cited By 0. doi:10.1109/CEC.2018.8477734.

URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85056273905&doi=10.1109%2fCEC.2018.8477734&partnerID=40&md5=630b5490cd7fb978ec75e7dcf69e3791

[46] Y. Bian, Z. Li, J. Guo, R. Zhao, Concrete hyperheuristic framework for test case prioritization, Journal of Software: Evolution and Process 30 (11), cited By 0. doi:10.1002/smr.1992.
URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85056445187&doi=10.1002%2fsmr.1992&partnerID=40&md5=65a4cc4982f65bfe8a857b7306436373

[47] H. Luiz Jakubovski Filho, T. Nascimento Ferreira, S. Regina Vergilio, Incorporating user preferences in a software product line testing hyper-heuristic approach, 2018, cited By 0. doi:10.1109/CEC.2018.8477803.
URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85056269713&doi=10.1109%2fCEC.2018.8477803&partnerID=40&md5=d5a11a2d07ff9c494c02c41a90f9e8e6

[48] H. Filho, T. Ferreira, S. Vergilio, Multiple objective test set selection for software product line testing: Evaluating different preference-based algorithms, 2018, pp. 162–171, cited By 0. doi:10.1145/3266237.3266275.
URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85055843164&doi=10.1145%2f3266237.3266275&partnerID=40&md5=cd6fb3a0c3023ab48b5531528183e822

[49] G. Guizzo, S. Vergilio, A pattern-driven solution for designing multi-objective evolutionary algorithms, Natural Computing (2018) 1–14Cited By 0; Article in Press. doi:10.1007/s11047-018-9677-y.
URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85044762810&doi=10.1007%2fs11047-018-9677-y&partnerID=40&md5=313bad97c8b7f445e59cc42bd505ef4c

[50] G. Guizzo, S. Vergilio, A. Pozo, Uma soluo baseada em hiper-heurstica para determinar ordens de teste na presena de restries de modularizao (2015) 9.

[51] A. Strickler, J. A. P. Lima, S. R. Vergilio, A. T. Pozo, Deriving products for variability test of feature models with a hyper-heuristic approach, Applied Soft Computing 49 (2016) 1232 – 1242. doi:https://doi.org/10.1016/j.asoc.2016.07.059.

[52] K. Z. Zamli, B. Y. Alkazemi, G. Kendall, A tabu search hyper-heuristic strategy for t-way test suite generation, Applied Soft Computing 44 (2016) 57 – 74. doi:https://doi.org/10.1016/j.asoc.2016.03.021.

[53] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. McMinn, An orchestrated survey of methodologies for automated software test case generation, Journal of Systems and Software 86 (8) (2013) 1978 – 2001. doi:https://doi.org/10.1016/j.jss.2013.02.061.

[54] S. U. R. Khan, S. P. Lee, R. W. Ahmad, A. Akhunzada, V. Chang, A survey on test suite reduction frameworks and tools, International Journal of Information Management 36 (6, Part A) (2016) 963 – 975. doi:https://doi.org/10.1016/j.ijinfomgt.2016.05.025.

[55] Y. Jia, M. B. Cohen, M. Harman, J. Petke, Learning combinatorial interaction test generation strategies using hyperheuristic search, in: Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE '15, IEEE Press, Piscataway, NJ, USA, 2015, pp. 540–550.

[56] T. do Nascimento Ferreira, J. N. Kuk, A. Pozo, S. R. Vergilio, Product selection based on upper confidence bound moea/d-dra for testing software product lines, in: 2016 IEEE Congress on Evolutionary Computation (CEC), 2016, pp. 4135–4142. doi:10.1109/CEC.2016.7744315.

[57] J. Kim, J. Park, E. Lee, A new hybrid algorithm for software fault localization, in: Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication, IMCOM '15, ACM, New York, NY, USA, 2015, pp. 50:1–50:8. doi:10.1145/2701126.2701207.

[58] D. J. Mala, V. Mohan, Quality improvement and optimization of test cases: A hybrid genetic algorithm based approach, SIGSOFT Softw. Eng. Notes 35 (3) (2010) 1–14. doi:10.1145/1764810.1764824.

[59] E. M. Fredericks, An empirical analysis of the mutation operator for runtime adaptive testing in self-adaptive systems, in: Proceedings of the 11th International Workshop on Search-Based Software Testing, SBST '18, ACM, New York, NY, USA, 2018, pp. 59–66. doi:10.1145/3194718.3194726.

[60] S. Cha, S. Hong, J. Lee, H. Oh, Automatically generating search heuristics for concolic testing, in: Proceedings of the 40th International Conference on Software Engineering, ICSE '18, ACM, New York, NY, USA, 2018, pp. 1244–1254. doi:10.1145/3180155.3180166.

[61] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: Nsga-ii, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197. doi:10.1109/4235.996017.

[62] M. Harman, S. A. Mansouri, Y. Zhang, Search-based software engineering: Trends, techniques and applications, ACM Comput. Surv. 45 (1) (2012) 11:1–11:61. doi:10.1145/2379776.2379787.

[63] T. Y. Chen, F.-C. Kuo, R. G. Merkel, T. Tse, Adaptive random testing: The art of test case diversity, Journal of Systems and Software 83 (1) (2010) 60 – 66, sI: Top Scholars. doi:https://doi.org/10.1016/j.jss.2009.02.022.
URL http://www.sciencedirect.com/science/article/pii/S0164121209000405

[64] M. Khatibsyarbini, M. A. Isa, D. N. Jawawi, R. Tumeng, Test case prioritization approaches in regression testing, Inf. Softw. Technol. 93 (C) (2018) 74–93. doi:10.1016/j.infsof.2017.08.014.

[65] K. McClymont, E. C. Keedwell, Markov chain hyper-heuristic (mchh): An online selective hyper-heuristic for multi-objective continuous problems, in: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11, ACM, New York, NY, USA, 2011, pp. 2003–2010. doi:10.1145/2001576.2001845.
URL http://doi.acm.org/10.1145/2001576.2001845

[66] Y. Zhang, M. Harman, G. Ochoa, G. Ruhe, S. Brinkkemper, An empirical study of meta- and hyper-heuristic search for multi-objective release planning, ACM Trans. Softw. Eng. Methodol. 27 (1) (2018) 3:1–3:32. doi:10.1145/3196831.
URL http://doi.acm.org/10.1145/3196831

[67] D. Jakobović, L. Jelenković, L. Budin, Genetic programming heuristics for multiple machine scheduling, in: M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, A. I. Esparcia-Alcázar (Eds.), Genetic Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 321–330.

[68] R. Poli, J. Woodward, E. K. Burke, A histogram-matching approach to the evolution of bin-packing strategies, in: 2007 IEEE Congress on Evolutionary Computation, 2007, pp. 3500–3507. doi:10.1109/CEC.2007.4424926.

[69] M. Maashi, E. zcan, G. Kendall, A multi-objective hyper-heuristic based on choice function, Expert Systems with Applications 41 (9) (2014) 4475 – 4493. doi:https://doi.org/10.1016/j.eswa.2013.12.050.
URL http://www.sciencedirect.com/science/article/pii/S095741741400013X

[70] W. Li, E. zcan, R. John, Multi-objective evolutionary algorithms and hyper-heuristics for wind farm layout optimisation, Renewable Energy 105 (2017) 473 – 482. doi:https://doi.org/10.1016/j.renene.2016.12.022.
URL http://www.sciencedirect.com/science/article/pii/S0960148116310709

[71] E. Zitzler, M. Laumanns, L. Thiele, Spea2: Improving the strength pareto evolutionary algorithm 103.

[72] P. Cowling, G. Kendall, E. Soubeiga, A hyperheuristic approach to scheduling a sales summit, in: E. Burke, W. Erben (Eds.), Practice and Theory of Automated Timetabling III, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 176–190.

[73] W. Li, E. Özcan, R. John, A learning automata-based multiobjective hyperheuristic, IEEE Transactions on Evolutionary Computation 23 (1) (2019) 59–73. doi:10.1109/TEVC.2017.2785346.

[74] H. Hemmati, A. Arcuri, L. Briand, Achieving scalable model-based testing through test case diversity, ACM Trans. Softw. Eng. Methodol. 22 (1) (2013) 6:1–6:42. doi:10.1145/2430536.2430540.
URL http://doi.acm.org/10.1145/2430536.2430540

[75] R. A. Oliveira, U. Kanewala, P. A. Nardi, Chapter three - automated test oracles: State of the art, taxonomies, and trends, Vol. 95 of Advances in Computers, Elsevier, 2014, pp. 113 – 199. doi:https://doi.org/10.1016/B978-0-12-800160-8.00003-6.
URL http://www.sciencedirect.com/science/article/pii/B9780128001608000036

[76] G. Fraser, A. Zeller, Mutation-driven generation of unit tests and oracles, IEEE Transactions on Software Engineering 38 (2) (2012) 278–292. doi:10.1109/TSE.2011.93.

[77] L. Thi My Hanh, T. Khuat, B. Nguyen, Mutation-based test data generation for simulink models using genetic algorithm and simulated annealing 03 (2014) 763–771.

[78] K. Ayari, S. Bouktif, G. Antoniol, Automatic mutation test input data generation via ant colony, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO '07, ACM, New York,

NY, USA, 2007, pp. 1074–1081. doi:10.1145/1276958.1277172.
URL http://doi.acm.org/10.1145/1276958.1277172

[79] L. Chung, J. C. S. do Prado Leite, On Non-Functional Requirements in Software Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 363–379. doi:10.1007/978-3-642-02463-4_19.
URL https://doi.org/10.1007/978-3-642-02463-4_19

[80] W. Afzal, R. Torkar, R. Feldt, A systematic review of search-based testing for non-functional system properties, Information and Software Technology 51 (6) (2009) 957 – 976. doi:https://doi.org/10.1016/j.infsof.2008.12.005.
URL http://www.sciencedirect.com/science/article/pii/S0950584908001833