Search-based Optimization for the Testing Resource Allocation Problem: Research Trends and Opportunities

Roberto Pietrantuono and Stefano Russo
Università degli Studi di Napoli Federico II
80125 Napoli, Italy
roberto.pietrantuono/stefano.russo@unina.it

ABSTRACT

This paper explores the usage of search-based techniques for the Testing Resource Allocation Problem (TRAP). We focus on the analysis of the literature, surveying the research proposals where search-based techniques are exploited for different formulations of the TRAP. Three dimensions are considered: the model formulation, solution, and validation. The analysis allows to derive several observations, and finally outline some new research directions towards better (namely, closer to real-world settings) modelling and solutions, highlighting the most promising areas of investigation.

CCS CONCEPTS

• **Software and its engineering → Software reliability**; **Software testing and debugging**; **Search-based software engineering**; *Software design engineering*;

KEYWORDS

Testing Resource Allocation, Reliability Allocation, Search-based Software Testing, Test Prioritization

ACM Reference Format:
Roberto Pietrantuono and Stefano Russo. 2018. Search-based Optimization for the Testing Resource Allocation Problem: Research Trends and Opportunities. In *SBST'18: IEEE/ACM 11th International Workshop on Search-Based Software Testing , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3194718.3194721

1 INTRODUCTION

Testing is the predominant factor of cost of software development. Engineers are demanded for delivering high-quality products on time and working with underestimated resources. Hence, test managers strive to use available resources in the best possible way. The need for cost-effective test planning led researchers to address this problem by systematic methods. *Testing resource allocation* is a planning procedure to assign testing resources to software components so as to achieve a target goal under given constraints. A lot of research effort has been spent to formulate models to optimize the resources allocation. These allows distributing resources on a quantitative basis, and not merely relying on the intuition or experience of test/project managers.

In recent years, as size and complexity of software systems increased and multiple objectives are required to be balanced to get optimal trade-offs (e.g., cost, schedule, reliability), the usage of search-based strategies to solve these problems spread out.

In this paper, we survey existing studies using search techniques to solve the Testing Resource Allocation Problem (TRAP). We analyze 17 studies with respect to several dimensions and highlighted current research trends in this area with respect to TRAP models formulation (Section 4), to search-based techniques adopted for models solution (Section 5) and to the model validation approach (Section 6). We then highlighte new research opportunities toward models and solutions better aligned with modern software development contexts (Section 7).

2 THE TESTING RESOURCE ALLOCATION PROBLEM

The problem of identifying those parts of a system[1] that should receive more resources during testing has been addressed by many researchers [19]. Typically, testing aims to detecting as many defects as possible, and testing resource allocation aims at distributing resources among modules so as to detect more defects with less effort [24]. Therefore, most of criteria allocate a greater effort to software modules expected to contain more defects.[2] To this aim, much research focused on *defect-proneness models*, i.e., models able to estimate the defectiveness of software modules based on process or product metrics. The output of these models are either the list of modules believed to be defect-prone (namely, containing at least one fault) and defect-free – in a binary classification formulation – or the rank of modules from the most to the least defect-prone one – when, more rarely, formulated as a ranking problem. The problem with this formulation is that the output does not directly tell how to allocate testing resources to modules, i.e., there is no allocation scheme following the classification/ranking task. Clearly, defect-prone modules (or the highest in the rank) deserve more testing resources, but the amount of resources is not quantified (apart from few exceptions, e.g. [1]).

SBST'18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5741-8/18/05…$15.00
https://doi.org/10.1145/3194718.3194721

[1] Here we refer to a *component* or *module* as an independently testable functionality. The terms are used as synonymous if not differently specified.
[2] Note that the software engineering literature typically uses the term *defect*, while in reliability and TRAP research the term *fault* is used, according to the *fault-error-failure chain* definition [3]: in this work we use them synonymously to denote the adjudged cause of an observed failure.

To properly quantify resources to distribute to modules, researchers formulated the TRAP as an optimization problem. The objective, in this case, is more refined, since the effort allocation criterion does not merely look for modules with more defects, but for the ones with a greater impact on the overall user-perceived reliability.[3] Reliability measure deals not only with *how many* defects are in the software, but also with *how often* they are activated and manifest themselves as failure. Thus, a software module with more defects than another can be more reliable inasmuch as those defects are less often activated and cause less frequent failures in operation [8]. TRAP models have *reliability improvement* as a main optimization objective, often contrasted with, or constrained on, testing *cost*, testing *effort* or testing *time* objectives – more details in the next Section.

The general form of a single-objective TRAP model looks like one of the following:

$$\begin{array}{ll} max!\ R_S(x|T_1,T_2,\ldots,T_n) & s.t.\ \ T = sum_{i=1}^{n} T_i \leq T^* \\ min!\ T = \sum_{i=1}^{n} T_i = & s.t.\ R_S(x|T_1,T_2,\ldots.T_n) \geq R^* \\ min!\ C(T_1,T_2,\ldots,T_n) & s.t.\ R_S(x|T_1,T_2,\ldots.T_n) \geq R^* \\ & \sum_{i=1}^{n} T_i \leq T^* \end{array} \quad (1)$$

where:

- R_S is the overall reliability of the software (or related functions, e.g., expected failure intensity, that is number of failure per time unit);
- T is the total testing effort, and
- T_i is the effort allocated to module i (out of n modules);
- C is a cost function dependent on testing efforts;
- T^* is the maximum available testing effort to distribute;
- R^* is the minimum level of required reliability.

The multi-objective version just considers two or more of these objectives together.

These functions are interdependent. For instance, the achieved reliability of a module depends on the testing effort devoted to it; the cost depends on testing effort (e.g., including the cost of tester) but can also account for the cost of debugging during testing or during operation. Thus, models vary based on the adopted objective function. Moreover, there are different ways of expressing inter-module architectural dependencies, which determine how an attribute of interest at system level (e.g., reliability) is computed from the same attribute at module level.

Besides formulation, the variety of solution techniques gives raise to further alternatives. Indeed, traditional optimal TRAP advocates the usage of exact methods (e.g., non-linear programming, dynamic programming). However, for complex system configurations and/or for handling multiple contrasting objectives, (meta)heuristic approaches are needed. Thus, depending *on how each objective function is constructed*, on *how many of them are considered*, on *how the architecture is described in the model*, and *on the solution method*, several TRAP variants are possible. Figure 1 provides a taxonomy of TRAP models.

[3]Reliability, in this research area, is defined as the probability of failure-free operation of a computer program for a specified time in a specified environment [22].

In the next Section, the focus is on the subset of problems solved by metaheuristics (i.e., excluding exact methods), so as to figure out how researchers applied search techniques to different variants of TRAP.

3 SEARCH-BASED TECHNIQUES FOR TRAP: RESEARCH RESULTS

The research considered an initial set of 188 papers, extracted by a keyword-based search on the *software testing resource allocation* problem followed by a manual filtering, conducted on the *SciVerse Scopus*[4] and the *IEEE Xplore*[5] search engines. From the abstract, most of them (114 papers) were filtered out, because the papers were note not directly related to the allocation problem (19 of them were on software reliability growth models; 31 on fault/change prediction techniques; 64 on allocation strategies unrelated with testing resources), getting to 74 papers dealing with TRAP. Out of this, we eliminated the papers that just formulate the TRAP problem (without discussing the solution method) and those (mostly, single-objective) adopting *exact methods*. The final set of selected primary studies solving the TRAP via a search-based technique includes 17 papers, ranging from 2003 to 2017.[6]

Figure 2 shows the distribution of the final set of studies by year and by publication venue – conference proceedings or journal. On this set, we analyzed *i)* the model formulation, *ii)* the model solution, and *iii)* the model validation.

[4]http://www.scopus.com
[5]http://ieeexplore.ieee.org/
[6]The final set of papers are References [2], [9], [11], [12], [16], [17], [18], [21], [23], [25], [27], [28], [29], [30], [31], [33], [34].

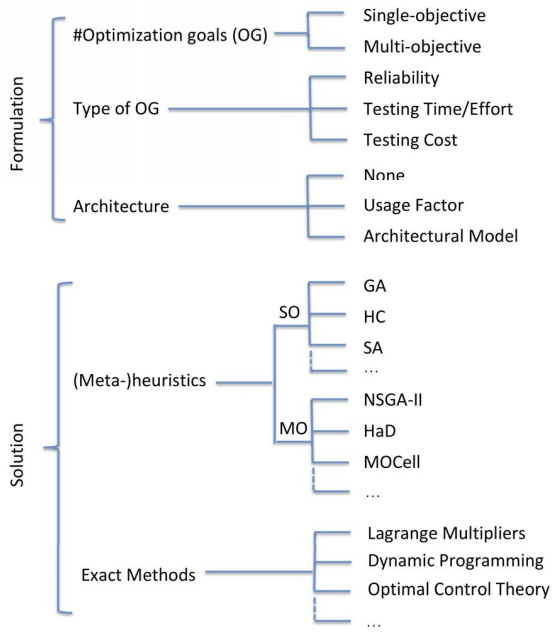

Figure 1: Classification of Testing Resource Allocation Problems (TRAPs)

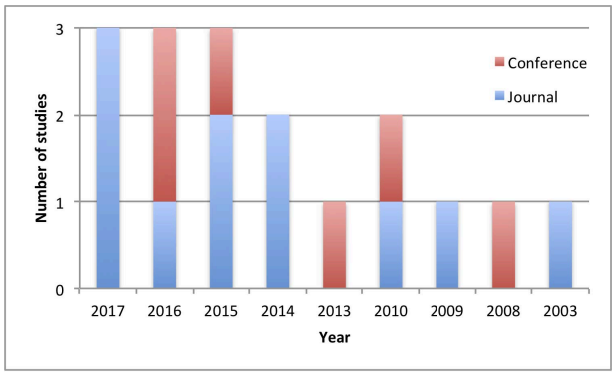

Figure 2: Search-based TRAP studies per year

Table 1: Search-based TRAP studies by *Formulation*

Objective Function and Architecture	Frequency
Objective: Obj. Function*	*n* out of 17
Reliability (R)	10
Time/Effort (T)	7
Cost (C)	15
Objective: SO vs MO	
Single-objective	8†
Multi-objective	9
Architecture Model	
None	9
Usage factor	1
Parallel-series model	6
Architecture-based (Markovian) model	1

* *The sum of occurrences is greater than 17, as multi-objective considers more of them together.*
† *Considered objectives in the single-SO case are: C: 6 times out of 8; R and T: 1 out of 8 (in the same article); R combined with C: 1 out of 8*

4 MODEL FORMULATION

TRAP models are distinguished based on the number (*single-* or *multi-* objective) and type of considered objectives, on how they describe the architecture in terms of inter-module relations (e.g., by usage factors, by parallel-series structures [19], by architectural models [26]). Table 1 shows the result of the analysis with respect to the model formulation, discussed in the following Sections.

4.1 Objectives

Looking at the general formulation (Eq. 1), the goals of interest for a TRAP formulation are: *reliability, testing time/effort, cost*. One or more of them can be set as objective or as constraint, obtaining several different formulations. The first part of Table 1 reports the number of studies considering reliability, testing time/effort and cost as an optimization goal, in both a single-objective (SO) formulation and in a multi-objective (MO) formulation.

Most of works (**10 out of 17**) include a reliability function as maximization objectives. In few cases **(3 out of 17)** it is considered as a lower-bound constraint. Most of solutions exploit *Software Reliability Growth Models* to capture the relation between the reliability growth (in terms of fault detection/correction process) and the testing effort (or time) devoted to each module. SRGMs are a wide class of models fitting inter-failure observed during testing and debugging to predict the next time to failure. The general form of the most common class of SRGMs, the Non-Homogeneous Poisson Process (NHPP) SRGMs, is:

$$m(t) = a\left(1 - e^{-D(t)}\right) \tag{2}$$

where:

- $m(t)$ is the cumulative number of detected and corrected faults at time t;
- $D(t) = \int_0^t \lambda(s)ds$ models the Fault Detection Process (FDP) by means of the *fault detection rate per remaining fault* (or failure intensity) λ;
- a is the expected number of total faults.

The shape of $D(t)$ determines the specific SRGM (e.g., exponential, S-Shaped, Logarithmic, Log-Logistic).

Equation 2 captures the FDP but neglects the *fault correction process* (FCP), namely it assumes an *immediate* debugging time, which can have a severe impact on the estimation accuracy [6], [7]. Researchers developed debug-aware SRGMs, which are more rarely integrated into TRAP models. The general debug-aware form is [20]:

$$m(t) = e^{-C(t)}\left(\int_0^t ac(s)e^{C(s)}[1 - e^{-D(s)}]ds\right) \tag{3}$$

where $m(t)$ now represents the cumulative number of detected and *corrected* faults, and $C(t) = \int_0^t \mu(s)ds$ models the FCP by means of the *fault correction rate per detected but not corrected fault*, μ. Thus, the objective to maximize is the sum, over each module, of $m(t)$ or a function thereof.[7]

Testing time or effort is the second objective. It appears as objective in **7 out of 17** cases, while it is almost always (except 1 case) a constraint (because it represents the available budget – in terms of time or effort – to not overcome). Rarely, models account for the relation between the *testing effort* (in terms of man-power) and *testing time* (in terms of calendar time, CPU time or number of test cases). In fact, while the simpler case assumes the testing effort E varying linearly with time T, this is, in general, not true. In the literature, the relation is modeled by the so-called *Testing Effort Functions* (TEFs), which model such a non-linearity by a function F: $E = F(T)$ – the most common TEF, shown to well represent the usual trend of testing effort, is the logistic TEF [14], [13], [15]. When considering a TEF, $m(t)$ in Eq. 2 and Eq. 3 change, since the effort Y is considered in lieu of time, making it more complex to solve. **4 out of 17** search-based TRAP models exploit a TEF to account for this difference.

Finally, almost all models (**15 out of 17**) account for *cost* as objective. Cost is a measure related to the effort spent, but goes beyond it. There are various cost models; a common model is the following one [14], [32]:

$$C(t) = C_1^* \cdot (\delta/24) \cdot m_c(t) + C_2^* \cdot (\delta/24) \cdot (m_d(\infty) - m_c(t)) + C_3^* \cdot (Y/24) \tag{4}$$

where:

[7] E.g., the failure intensity $\lambda(t)$, or the expected reliability at operational time t, expressed as $R(t) = exp(-\lambda(T) \cdot t)$, where $\lambda(T)$ is the failure intensity at the end of testing (at time T) assuming no change in the software during operation [26]).

C_1^* is the cost per man-day to correct a fault during testing;
C_2^* is the cost per man-day to correct a fault at runtime (typically $C_2^* \geq C_1^*$ [4]);
C_3^* is the cost per testing-effort expenditure unit (e.g., man-day), i.e., hourly or daily cost of a tester;
δ is the average number of hours to fix a fault.

As for the number of objectives, early works focused on *single-objective* (SO) optimization. Examples are the works by Kapur *et al.* [16], who consider the cost as objective, solved by a genetic algorithm (GA), by Dai *et al.* [9], who formulate a combined reliability and cost function, solved by a GA, and by Aggarwal et al. [2], who maximize reliability (and then, minimize effort in a second formulation) again using a GA. if we refer to the whole set of TRAP models (including the ones solved by exact methods or with no solution algorithm), single-objective works are indeed the vast majority. However, restricting the scope to works solved by search techniques, there is a balance, since multi-objective (MO) problems are difficult to handle without search techniques: **8 papers** formulate a single-objective problem, while **the remaining 9** deal a multi-objective one.

An example of MO TRAP is by Huang *et al.* [33], who take the three objectives together, with cost described by Eq. 4, testing time with no TEF and a general SRGM, and also consider the relation among modules (i.e., architecture) explicitly in the formulation; Sangeetha *et al.* [27] use an exponential SRGM and a problem-specific const function under effort constraint; Pietrantuono et *al.* [25] consider a general TEF-aware testing time function, a debug-aware SRGM, and a cost function as in Eq. 4, also considering the uncertainty of parameters to get a robust solution. To summarize, two remarks follow.

Observation 1
Considering the aggregate count (without distinguishing single-objective and multi-objective formulations), a Cost function is present in almost all the cases, the Reliability (10 cases) and testing time/effort (in 7 cases); in single-objective formulations (8 out of 17 cases), cost is again almost always present as optimization goal (except one case), reliability and testing time/effort appear only in 1 paper, where both formulations are proposed in sequence. Thus, cost is indeed deemed the most important optimization goal, under testing time/effort and/or reliability constraints.

Observation 2
Testing time/effort is considered just once in single-objective optimization and 7 times in multi-objective setting (the least considered one); on the other hand, it is set as a constraint in 16 out of 17 studies – hence the formulations represent the most common situation where a given budget is available and should not be overcome). A refinement in modeling the testing time/effort is considered just in 4 cases, wherein a TEF is used to model the time-effort relation.

4.2 Architecture Model

While SRGMs describe the reliability-testing relation of a single module, the relation between modules should be also taken into account. In fact, the allocation process should also consider how often a module is used – e.g., if the total reliability computation does not account for module usage, a solution can assign a lot of testing resources to a rarely used module, which will not actually contribute to the total reliability, thus wasting resources. About half of the works (**9 out of 17**) do not account for any form of inter-module relation. A solution, often adopted by TRAP models, but rarely seen (only 1 case) for *search-based* TRAP models, is to consider a usage factor – a [0-1] weight expressing the frequency (or probability) with which a module will be involved [25].

Another solution is to consider structures of series-parallel (or more complex, like bridged, star) and the associated structure function [19], [34]. This approach is taken from the literature on Redundancy Allocation Problem (RAP) – the one to determine at design time the minimum required component-level reliability against a system-level reliability objective. In search-based TRAPs, this was adopted in **6 cases**.

A more complete solution is to consider Markovian architectural models, like Discrete-time Markov Chains (DTMC), representing modules by states, with transition probabilities describing the execution flow from one component to another [10], [26]. The derived measure, such as the so-called *visit count* (i.e., the average number of visits to a component), is used in the system-level reliability computation.

Observation 3
Despite its practical importance to obtain results close to the reality, the architecture is often not considered in the TRAP models. When considered, it is accounted for by means of parallel-series models, which however are more appropriate for redundancy allocation problem.

5 MODEL SOLUTION

17 out of 74 papers (that is **23%**) consider metaheuristics to solve the allocation problem, because the complexity of the formulated model would lead to unacceptable computational time by an exact method solution. Table 2 lists the metaheuristics used in the selected studies and how many times they performed better than other competing metaheuristics.

Observation 4
A good variety of metaheuristics have been tried to solve the TRAP problem – 12 MO and 7 SO metaheuristics in 17 papers. The most used ones are (variants of) the NSGA-II (Non-dominated Sorting Genetic Algorithm) followed by the HaD (Harmonic Distance) algorithm for multi-objective problems and the GA for single-objective problems (used 6 times).

Table 2: Search-based TRAP studies by *Solution*

Search Algorithm: Multi-Objective^{+}	**Frequency**	**Best-performing**$^{\diamond}$
NSGA-II	7	3 times out of 7 comparisons
HaD	4	1 out of 5
MODE	2	1 out of 1
WNS-MODE	2	1 out of 2
PAES	2	0 out of 2
MOCELL	2	0 out of 2
CELLDE	1	0 out of 1
MOEA/D	1	1 out of 1
IBEA	1	0 out of 1
RWGA	1	1 out of 1
SPEA2	1	0 out of 1
SMPSO	1	0 out of 1
Search Algorithm: Single-Objective^{+}	**Frequency**	**Best-performing**$^{\diamond}$
GA	6	1 out of 1
GLSA	1	1 out of 1
Hill Climbing	1	1 out of 1
Simulated Annealing	1	0 out of 1
K-A	1	0 out of 1
Y-X	1	0 out of 1
T-M	1	0 out of 1

$^{+}$ *Listed algorithms are either in their original version or variants* tailored to TRAP.
$^{\diamond}$ *Count the number of wins with respect to the number of comparisons: in a paper, there might more comparisons (e.g., on 2 and then 3 objectives), or no comparison at all.*

Other popular algorithms for the multi-objective cases are those based on Differential Evolution, namely Multi-Objective Differential Evolution (MODE), Weighted Normalized Sum-Multi-Objective Differential Evolution (WNS-MODE) and the hybrid Cellular Differential Evolution (Cell-DE), combining Multi-objective Cellular (MOCell) and MODE, which account for 5 cases altogether. Furthermore, widely-used metaheuristics are tried for TRAP, like Pareto Archived Evolution Strategy (PAES), Multi-objective Particle Swarm Optimization (SMPSO), and Indicator-based Evolutionary Algorithm (IBEA).

For the single-objective case, other algorithms, besides the conventional GA, are: a variant of GA, the Genetic Local Search Algorithm (GLSA), combining GA with local search; the well-known Hill Climbing (HC), Simulated Annealing (SA) and other problem-specific algorithms (K-A, Y-X, T-M, from initials of authors' name).

In the multi-objective case, there is a wider variety of approaches (unlike the SO case); the prevalence of NSGA-II is not necessarily because it performs (or is expected to perform) better, but because of its popularity and because TRAP studies focus more on proposing new models than on finding the best metaheuristic. This hypothesis is corroborated by looking at the number of times in which NSGA-II performs better than others – a detail reported in Column 3 of Table 2.

Specifically, most of the studies compare several metaheuristics according to either conventional coverage, convergence and diversity metrics (e.g., hypervolume, (inverse) generational distance, spread) or using problem-specific metrics (i.e., which algorithm yields the best solution, or the best cost-optimality trade-off, for a given objective function). From comparisons, the following remark emerges.

Observation 5
In the MO case, NSGA-II performs better than competing metaheuristics in 3 out of 7 cases. It is the most used algorithm, but in 4 cases it is not the best one. HaD loses a comparison in 4 out of 5 cases. Algorithms based on DE (MODE, WNS-MODE) wins a comparison in 2 out of 3 cases, while the hybrid CellDE looses 1 out 1 times. PAES, IBEA, SPEA2, SMPSO, and MOCell lose their comparisons (7 cases in total). MOEA/D, RWGA and MODE are compared 1 time (in different studies) and win the comparison against the others.

In particular, NSGA-II is defeated: by the Random-Weighted Genetic Algorithm (RWGA) [29] in a comparison with 6 other metaheuristics – hence, RWGA performance is worth to be highlighted; by the Multi-objective Evolutionary Algorithm based on Decomposition (MOEA/D) [28] (both coupled with a local search strategy); by the HaD [31] in a 3-objective comparison case; by MODE in [30]. On the other hand, it won the comparison against MOCell, PAES and IBEA in [25]; a variant of NSGA-II for TRAP won against WNS-MODE and HaD in [34]; NSGA-II won against HaD in [31] in a 2-objective comparison.

It is worth to highlight the good performance of DE-based algorithms (they loose 1 out of 3 against a problem-specific variant of NSGA-II), the average performance of NSGA-II, and the bad performance of several older metaheuristics. It is also worth to mention less usual algorithms MOEA/D and RWGA, which won 1 out of 1 comparison (the former against NSGA-II and HaD, the latter against NSGA-II, CellDE, PAES, SPEA2, SMPSO, MOCEL). The following two further observations are about SO algorithms and local search strategies.

Observation 6
In the SO case, there are less cases where a comparison is carried out. GA performed better than Y-X and T-M algorithms [11], GLSA outperforms Simulated Annealing an K-A algorithms, while Hill Climbing outperforms a greedy-based policy.

Observation 7
In the two cases in which it is used, the effect of local search was found to be beneficial [11], [28]. The former reports the formulation of the discussed GLSA variant of GA in a single-objective setting; while the latter adopts a local search strategy to improve multi-objective metaheurstics: MOEA/D, HaD, NSGA-II.

6 MODEL VALIDATION

Table 3 shows the number of times in which a study has been validated by numerical examples or by an experimental study and/or in an industrial context. The distinction refers to how the dataset,

on which metaheuristics are tested, is derived and on the considered system under test: in a validation by numerical examples, the dataset is generated artificially for the purpose of illustration or taken from the literature; in experimental validation, a randomized experiment or a quasi experiment is conducted, and it can be in an industrial setting or not (e.g., on open source software).

Observation 8
The vast majority of studies on search-based TRAP use numerical illustrations to validate their models. Typically, one or more metaheuristics are tested on numerical examples, and the system under test is often an artificial example too (especially in the case of parallel-series systems). Only two cases test their approach in industrial settings – one at Cisco Norway, on a videoconferencing system (VCS) [29], and the other at an Healthcare company, on an CRM software [25] – and one on open source software [12].

7 RESEARCH DIRECTIONS

The observations resulting form exploring the state of the art inspires new research directions. We mention the most relevant ones:

- **TRAP objective functions**. Both SRGMs for reliability objective as well as effort and cost models, are notoriously subject to numerous assumptions. Despite 40 years of research on SRGMs, there is still the need to work on new and more favourable models reflecting the current testing (and debugging) practices to capture the real reliability growth trend.
  Similarly, there is wide room for improvement in the formulation of effort and cost models more closely representing real processes. With multi- and many-objectives search techniques, much more can be done in terms of resource allocation optimization by more refined models tailored to current production processes, e.g.: including separate cost or effort goals for debugging and testing processes (e.g., testers/debuggers by skills, experience); including bug-debugger assignment policies (e.g., select debugger based on bug severity) or functionality-testers assignment policies (e.g., select tester based on functionalities to test).
  Other aspects to model include the *human factors* involved in a testing/debugging process, which is a wide research area too, and other attributes of interest besides reliability, such as *power consumption*, *security* and *performance*.
- **TRAP in new contexts**. A concrete realization of the above research direction is to devise TRAP models for emerging software production contexts, such as agile and DevOps processes, where the rapid release cycles allow for exploiting runtime data as feedback for building models closer to the system-in-operation. An optimal allocation of testing resources could take place at each cycle leveraging data of previous ones.
  Of course, this research area entails revisiting objective functions too. For instance, new SRGMs (e.g., based on SRGMs with multiple change points) or new ways to model the testing-reliability relation for these new processes are needed. Similarly, the description of the architectural relation among modules is no longer static, at design time, but applications are loosely coupled and dynamically interact with each other at runtime (e.g., web services, microservices). Hence, architectures and behavioural models inference at runtime (exploited for re-allocating resources from time to time) becomes a crucial research area.
- **TRAP under uncertainty**. Models are naturally affected by uncertainty. While it is undebatable that uncertainty should be reduced (e.g., by more accurate models, as discussed above), some uncertainty remains inevitable in software-related processes. Hence, besides *uncertainty reduction*, *uncertainty tolerance* becomes important.
  Robust optimization is one way of dealing with that, as it considers the uncertainty associated with models' parameters (e.g., in terms of confidence interval) and provides *ranges* of solutions (i.e., interval-solutions instead of point-solutions) under different possible values of such parameters. An example is in our previous work [25]. A further possibility is to adopt a dynamic strategy, namely to solve the TRAP model more times as the testing proceed.
  With time, the impact of upfront assumptions becomes clear, and the solution becomes adaptive with respect to the real runtime context (e.g., SRGMs becomes more and more accurate as faults are detected) [5], [21]. This leads to a new research direction aimed at developing/applying dynamic search-based metaheuristics to TRAP models.
- **TRAP in the real world**. For TRAP models to increase their popularity and usefulness, much more real-world experiences are needed, as resulting from Section 6. Applying models to concrete instances, especially in industry, allows eliciting new requirements and testing assumptions to figure out which ones are realistic and which call for new, more comprehensive, models. To this end, the support of tools would also be paramount.

Table 3: Search-based TRAP studies by *Validation*

Validation Method	Frequency
Numerical	14
Experiment and/or Industry application	3

8 CONCLUSIONS

As size and complexity of software systems and of related testing/debugging process increase, search-based strategies becomes increasingly important for the Testing Resource Allocation Problem. This work conducted an analysis on the existing literature on TRAP. The sample size of 17 papers do not allow for drawing general conclusions, but the work highlighted the main dimensions along which TRAP models are built and evolve, and which search techniques are being used for their solution. Then, it outlined possible directions to develop new and more accurate models, possibly more suitable for modern software current software development processes. The work finally drew four research directions, emphasizing the importance of both new formulations and of real-world

(industrial) experiences to advance state-of-the-art and state-of-the-practice together.

ACKNOWLEDGMENTS

This work has been partially supported by the GAUSS Italian research project, funded by MIUR under the PRIN 2015 program (Grant n. 2015KWREMX_002).

REFERENCES

[1] A. Monden et al. 2013. Assessing the Cost Effectiveness of Fault Prediction in Acceptance Testing. *IEEE Transactions on Software Engineering* 39, 10 (Oct 2013), 1345–1357.

[2] A.G. Aggarwal, G. Kaur, and P.K. Kapur. 2010. Optimal testing resource allocation for modular software considering imperfect debugging and change point using genetic algorithm. In *2nd International Conference on Reliability, Safety and Hazard - Risk-Based Technologies and Physics-of-Failure Methods (ICRESH)*. IEEE, 535–541.

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 1, 1 (Jan 2004), 11–33.

[4] B.W. Boehm. 1981. *Software Engineering Economics* (1st ed.). Prentice Hall PTR.

[5] G. Carrozza, R. Pietrantuono, and S. Russo. 2014. Dynamic test planning: a study in an industrial context. *International Journal on Software Tools for Technology Transfer* 16, 5 (Oct 2014), 593–607.

[6] M. Cinque, D. Cotroneo, A. Pecchia, R. Pietrantuono, and S. Russo. 2017. Debugging-workflow-aware software reliability growth analysis. *Software Testing, Verification and Reliability* 27, 7 (Nov 2017).

[7] M. Cinque, C. Gaiani, D. De Stradis, A. Pecchia, R. Pietrantuono, and S. Russo. 2014. On the Impact of Debugging on Software Reliability Growth Analysis: A Case Study. In *Computational Science and Its Applications – ICCSA 2014 (LNCS)*, Murgante, B. et alii (Ed.), Vol. 8583. Springer International Publishing, 461–475.

[8] D. Cotroneo, R. Pietrantuono, and S. Russo. 2013. Combining Operational and Debug Testing for Improving Reliability. *IEEE Transactions on Reliability* 62, 2 (2013), 408–423.

[9] Y.S. Dai, M. Xie, K.L. Poh, and B. Yang. 2003. Optimal testing-resource allocation with genetic algorithm for modular software systems. *Journal of Systems and Software* 66, 1 (Apr 2003), 47–55.

[10] L. Fiondella and S.S. Gokhale. 2012. Optimal Allocation of Testing Effort Considering Software Architecture. *IEEE Transactions on Reliability* 61, 2 (June 2012), 580–589.

[11] R. Gao and S. Xiong. 2015. A genetic local search algorithm for optimal testing resource allocation in module software systems. In *Intelligent Computing Theories and Methodologies (LNCS)*, D.-S. Huang et al. (Ed.), Vol. 9226. Springer International Publishing, 13–23.

[12] H. Hemmati, M. Nagappan, and A.E. Hassan. 2015. Investigating the effect of "defect co-fix" on quality assurance resource allocation: A search-based approach. *Journal of Systems and Software* 103 (May 2015), 412–422.

[13] C.-Y. Huang, S.-Y. Kuo, and M.R. Lyu. 2007. An Assessment of Testing-Effort Dependent Software Reliability Growth Models. *IEEE Transactions on Reliability* 56, 2 (June 2007), 198–211.

[14] C.-Y. Huang and J.-H. Lo. 2006. Optimal resource allocation for cost and reliability of modular software systems in the testing phase. *Journal of Systems and Software* 79, 5 (May 2006), 653–664.

[15] C.-Y. Huang and M.R. Lyu. 2005. Optimal release time for software systems considering cost, testing-effort, and test efficiency. *IEEE Transactions on Reliability* 54, 4 (Dec 2005), 583–591.

[16] P.K. Kapur, A.G. Aggarwal, K. Kapoor, and G. Kaur. 2009. Optimal testing resource allocation for modular software considering cost, testing effort and reliability using genetic algorithm. *International Journal of Reliability, Quality and Safety Engineering* 16, 06 (Dec 2009), 495–508.

[17] V. Kumar, P.K. Kapur, N. Taneja, and R. Sahni. 2017. On allocation of resources during testing phase incorporating flexible software reliability growth model with testing effort under dynamic environment. *International Journal of Operational Research* 30, 4 (2017), 523–539.

[18] V. Kumar, S.K. Khatri, H. Dua, M. Sharma, and P. Mathur. 2014. An assessment of testing cost with effort-dependent fdp and fcp under learning effect: a genetic algorithm approach. *International Journal of Reliability, Quality and Safety Engineering* 21, 06 (2014), 1450027.

[19] W. Kuo and R. Wan. 2007. Recent Advances in Optimal Reliability Allocation. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 37, 2 (Mar 2007), 143–156.

[20] J.-H. Lo and C.-Y. Huang. 2006. An integration of fault detection and correction processes in software reliability analysis. *Journal of Systems and Software* 79, 9 (2006), 1312–1323.

[21] Y. Lu, F. Yue, G. Zhang, Z. Su, and Y.-Q. Wang. 2016. Model and solution to testing resource dynamic allocation for series-parallel software systems. 27 (08 2016), 1964–1977.

[22] M.R. Lyu (Ed.). 1996. *Handbook of Software Reliability Engineering*. McGraw-Hill, Inc., Hightstown, NJ, USA.

[23] M. Nasar and P. Johri. 2016. Testing Resource Allocation for Fault Detection Process. In *Smart Trends in Information Technology and Computer Communications*, A. Unal et al. (Ed.). Springer, Singapore, 683–690.

[24] Mauro Pezzè and Michal Young. 2007. *Software Testing and Analysis: Process, Principles and Techniques*. Wiley.

[25] R. Pietrantuono, P. Potena, A. Pecchia, D. Rodriguez, S. Russo, and L. Fernandez. 2017. Multi-Objective Testing Resource Allocation under Uncertainty. *IEEE Transactions on Evolutionary Computation* PP, 99 (2017).

[26] R. Pietrantuono, S. Russo, and K.S. Trivedi. 2010. Software Reliability and Testing Time Allocation: An Architecture-Based Approach. *IEEE Transactions on Software Engineering* 36, 3 (May 2010), 323–337.

[27] M. Sangeetha, C. Arumugam, K.M. Senthil Kumar, and S. Hari Shankar. 2015. An Effective Approach to Support Multi-objective Optimization in Software Reliability Allocation for Improving Quality. *Procedia Computer Science* 47 (2015), 118–127.

[28] Y. Shuaishuai, F. Dong, and B. Li. 2013. Optimal Testing Resource Allocation for modular software systems based-on multi-objective evolutionary algorithms with effective local search strategy. In *IEEE Workshop on Memetic Computing (MC)*. IEEE, 1–8.

[29] S. Wang, S. Ali, T. Yue, Ø. Bakkeli, and M. Liaaen. 2016. Enhancing Test Case Prioritization in an Industrial Setting with Resource Awareness and Multi-objective Search. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. 182–191.

[30] Z. Wang, K. Tang, and X. Yao. 2008. A multi-objective approach to testing resource allocation in modular software systems. In *IEEE Congress on Evolutionary Computation*. IEEE, 1148–1153.

[31] Z. Wang, K. Tang, and X. Yao. 2010. Multi-Objective Approaches to Optimal Testing Resource Allocation in Modular Software Systems. *IEEE Transactions on Reliability* 59, 3 (Sept 2010), 563–575.

[32] S. Yamada, J. Hishitani, and S. Osaki. 1993. Software-Reliability Growth with a Weibull Test-Effort: A Model & Application. *IEEE Transactions on Reliability* 42, 1 (Mar 1993), 100–106.

[33] B. Yang, Y. Hu, and C.-Y. Huang. 2015. An Architecture-Based Multi-Objective Optimization Approach to Testing Resource Allocation. *IEEE Transactions on Reliability* 64, 1 (Mar 2015), 497–515.

[34] G. Zhang, Z. Su, M. Li, F. Yue, J. Jiang, and X. Yao. 2017. Constraint Handling in NSGA-II for Solving Optimal Testing Resource Allocation Problems. *IEEE Transactions on Reliability* 66, 4 (Dec 2017), 1193–1212.