**RESEARCH QUESTIONS**

\*RTSO = Regression Test Suite Optimization

|  |  |  |  |
| --- | --- | --- | --- |
| **RQs** | **RQ Statement** | **Motivation** | **Metrics** |
| **RQ 1** | ***What is the status of the research in RTSO approaches?*** | | |
| **RQ 1.1** | *What are the research trends in RTSO?* | Identify whether **interest** in the field is growing, stagnating, or declining, revealing **shifts** in methodological paradigms. | 1. Pub Year 2. Method: {Selection, Minimization, Prioritization} |
| **RQ 1.2** | *What are the similarities and differences between RTSO approaches?* | **Comparing** approaches through their goals, assumptions, and trade-offs, making **informed decisions** on which method to use based on the problem. | Taxonomy Class |
| **RQ 1.3** | *How are RTSO problems usually formalized?* | Shedding light on how well current techniques capture the **complexity** of real-world **constraints** and **objectives**. | {Multi-Objective,  Formalization  Single-Objective Formalization} |
| **RQ 2** | ***What types of SUT are used in RTSO experiments?*** | | |
| **RQ 2.1** | *What datasets/SUTs are the most used in RTSO techniques?* | Promoting **reproducibility** and highlighting potential **over-reliance** on specific datasets, which can threaten external validity. | Dataset Name |
| **RQ 2.2** | *What are the main characteristics of commonly used SUT?* | This question clarifies whether research targets **specific** **domains** or is **broadly applicable**, if proposed methods and tools solve **real-world scenarios** and so on. | 1. Application Domain 2. # of Test Cases 3. LOC |
| **RQ 3** | ***What criteria are used by RTSO techniques?*** | The criteria determine the optimization performance. Understanding the used features allows for a better **interpretation** of algorithms and **fair comparisons**. | {[statement/branch/…/ method] coverage, execution time, fault detection history, failure rate,...} + |
| **RQ 4** | ***How have empirical evaluations of RTSO techniques been conducted?*** | | |
| **RQ 4.1** | *What metrics are used to evaluate RTSO techniques?* | Investigating what the community values and helping detect **gaps** in **assessment** practices. | {Hypervolume, APFD, …} |
| **RQ 4.2** | *How is reproducibility ensured in RTSO research?* | Assessment of **reproducibility** in RTSO research. | 1.0: Full replication package (scripts+datasets+instructions)  0.5: Partial artifact or insufficient instructions  0.0: No code, no artifact available |

**TAXONOMY**

**Coverage-Based:** ranks test cases based on how much structural code coverage they provide. Variants include statement and branch coverage, and Fault Exposing Potential (FEP). These techniques are typically evaluated using the APFD metric, which quantifies how quickly faults are detected.

**Model-Based:** prioritizes/selects test cases using behavioral models of the system under test. When a model is updated due to software changes, test cases are classified into high-priority (TSH) if they are affected by the modification, and low-priority (TSL) otherwise. The initial strategy prioritizes TSH randomly, followed by TSL. More advanced versions of the approach incorporate dependency analysis, allowing prioritization to consider both direct and indirect impacts of changes within the model.

**Requirement-Based:** prioritizes/slects test cases according to the software requirements they cover. Each test case is linked to one or more requirements, and optimization is based on attributes of those requirements, such as customer-assigned priority or implementation complexity.

**Probability-Based:** uses statistical and probabilistic models to guide the ordering/selection of test cases. This approach can leverage execution history, selecting tests that have not been run recently (e.g., using Least Recently Used heuristics); it can use Bayesian Networks to estimate fault detection likelihood based on code changes and test coverage, dynamically updating probabilities during execution; some applies matrix decomposition (SVD) to identify clusters of frequently co-modified files, aligning them with test cases and recent code modifications.

**Distribution-Based:** Distribution-based techniques minimize and prioritize test cases based on the distribution of the profiles of test cases in the multi-dimensional profile space. Test case profiles are produced by the dissimilarity metric, a function that produces a real number representing the degree of dissimilarity between two input profiles. Using this metric, test cases can be clustered according to their similarities.

**Human-Based (Clustering):** integrates human intuition with machine learning. Testers provide pairwise comparisons between test cases, indicating which should be executed first. These preferences are used by a learning-to-rank algorithm to generate a prioritization/selection function. Initial heuristics like statement coverage and cyclomatic complexity support the learning process. **Human-Based (Non-Clustering):** To improve scalability, clustering is introduced in the Human-based approach: testers rank clusters instead of individual tests (inter-cluster), while tests within each cluster are ordered automatically (intra-cluster)

**History-Based:** prioritizes/selects test cases based on historical change patterns among software artefacts. Using Singular Value Decomposition (SVD) on a change matrix, it identifies association clusters—groups of files frequently modified together. Each file is linked to test cases that affect or execute it. When a new system change occurs, it is encoded as a modification vector, and test cases are prioritized/selected based on how closely they relate to the changed files via their cluster associations.

**Cost-Aware:** addresses resource constraints by considering test execution costs and fault severities. Cost-aware approaches optimize for limited budgets. Metrics like APFDc incorporate both cost and severity. Some techniques (e.g., time-aware prioritization) select a test subset that fits within a time limit using search-based methods like genetic algorithms. Others apply multi-objective optimization (e.g., Pareto-front) or ILP to balance trade-offs. These methods aim to detect severe faults early and efficiently, and are especially valuable when only partial testing is feasible due to time or resource limits.

**Others:** includes a variety of specialized techniques. Mutation-based targets interface contracts in component-based systems. Session-based uses real user sessions to test web applications, leveraging criteria like HTTP request count or 2-way parameter-value coverage. Property-based approaches prioritize/select tests that could violate model properties, as determined by model checkers. Data-flow and call-tree models rank tests by def-use pairs or call-path coverage, respectively. Slice-based methods prioritize/select tests with large or relevant program slices.

