

mSITG-MD: A Metadata-Aware Particle Swarm Optimization Framework for Combinatorial Test Data Generation in IoT and Multimedia Systems

Mohammad Naderuzzaman

142408@student.duet.ac.bd

Dhaka University of Engineering & Technology

Mohammad Abul Kashem

Dhaka University of Engineering & Technology

Nadiba Zaman Kaifa

Sonargaon University

Research Article

Keywords: Combinatorial testing, test-data generation, swarm intelligence, metadata modeling, Internet of Things, multimedia testing, Particle Swarm Optimization

Posted Date: October 19th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11209004/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

mSITG-MD: A Metadata-Aware Particle Swarm Optimization Framework for Combinatorial Test Data Generation in IoT and Multimedia Systems

Mohammad Naderuzzaman (<https://orcid.org/0009-0004-0180-8703>)^{1,*}, Mohammad Abul Kashem (<https://orcid.org/0009-0009-2055-6050>)², Nadiba Zaman Kaifa (<https://orcid.org/0009-0001-6508-2452>)³

^{1,2,3}*Department of Computer Science & Engineering*

^{1,2}*Dhaka University of Engineering & Technology, Gazipur, Bangladesh*

³*Sonargaon University, Dhaka, Bangladesh*

E-mail:142408@student.duet.ac.bd¹, drkashemll@duet.ac.bd², cse2403033039@su.edu.bd³

Abstract

Increasing adoption of the Internet of Things (IoT) and multimedia with the software systems has made software testing increasingly challenging because of their heterogeneous nature, complex parameter dependencies, large search spaces and stringent compatibility constraints. Existing test-data suite or test suite generation techniques generally treat configuration parameters uniformly, so they have limited capability to handle metadata characteristics that includes discrete, continuous, hierarchical and IoT-specific parameters. This paper presents a metadata-aware framework **mSITG-MD (modified Swarm Intelligence Test Generation for MetaData)** based on particle swarm optimization algorithm, for automated test-data generation in configurable IoT and multimedia systems. This framework integrates metadata-aware parameter modeling, constraint-aware particle initialization, a weighted fitness function for prioritizing critical metadata interactions, constraint validation and repair and dynamic search-space pruning in a single module. The approach is evaluated by using six benchmark systems and a realistic IoT-based video-processing system comprising 12 metadata input parameters and 22 constraints. Each experiment is repeated 30 times to account for the stochastic nature of swarm-based optimization. The experimental results of mSITG-MD produces test suites that are **4–9% smaller** than the best evaluated baseline methods for interaction strengths of $t = 3-6$, while achieving comparable performance at $t=2$ and achieves **21–28% lower generation time** than the baseline particle swarm approach for systems with larger configurations. Moreover, this framework generates **97% valid test data**, achieves **100% IoT compatibility coverage**, and satisfies **98% of the specified metadata constraints**. These performances indicate that incorporating metadata characteristics and domain-specific constraints into swarm intelligence-based test-data generation can improve test-suite's effectiveness, generation efficiency, and applicability to complex IoT and multimedia systems.

Keywords: Combinatorial testing, test-data generation, swarm intelligence, metadata modeling, Internet of Things, multimedia testing, Particle Swarm Optimization.

1. Introduction

Software testing remains fundamental and costly activity in software engineering, because the software failures may occur from complex interactions among parameters, input values and environmental conditions. Proper testing of software is still a challenge for the software industries because of the lack of availability of necessary testing tools. Moreover recent software development had to deal with the Internet of Things (IoT) and multimedia as integral part of software system [1]. Complex input parameters of these IoT and multimedia are not like conventional software, they combine devices with different capabilities, communication protocols, sensors, processing resources and software configurations [1, 2]. Existing test suite generation techniques are not designed to handle such situations. This paper addresses such problems with a metadata-aware swarm intelligence approach called **mSITG-MD** framework. This framework extends swarm-based test-data generation by explicitly modeling heterogeneous metadata, incorporating domain-specific constraints into test generation, assigning higher priorities to critical interactions, and dynamically reducing unproductive regions of the search space.

1.1 The Growing Challenge of Testing Modern Systems

Currently the IoT and multimedia systems have become fundamental and integral components of modern computing infrastructure [1]. These systems are characterized by their heterogeneous and decentralized nature with a massive data collection and processing capabilities [2]. A big smart parking system for instance, may consist of over 1,800 heterogeneous devices that includes sensors, servers, wireless access points, and user interfaces [3] for managing vehicles. Similarly, the multimedia systems process rich media with complex metadata and involve intricate parameter interactions, failing to test such systems can trigger failures [4].

Software testing, which typically consumes 40-50% of total software development budgets [5] remains the primary means of quality assurance. However, because of the combinatorial explosion problem, the exponential growth of test data required to exhaustively test modern configurable systems presents a formidable challenge [6]. Let us consider an example of a system with n parameters, each having v possible values, to exhaustively testing this system requires v^n test-data—a number that quickly becomes infeasible even for modest systems.

1.2 The Metadata Challenge

Metadata is a data that describes other data—is central to the functionality of the IoT and multimedia systems [7]. Unlike all the traditional software inputs, metadata input parameters exhibits three distinctive characteristics that make test-data generation particularly challenging:

Heterogeneity: Metadata can be (i) discrete (e.g. Container: {MP4, AVI, MKV}), (ii) Continuous (e.g. Bitrate: 500-10000 kbps), and (iii) Hierarchical (e.g. Profile depends on Codec) [4]. This means, unlike traditional data that is typically homogeneous in nature, metadata encompasses diverse types and structures.

Complex Constraints: A technical specification of any metadata imposes strict rules on its combinations. For example, HEVC video codec cannot be stored in an AVI container [8] because of incompatibility. In IoT the systems, certain devices may not be compatible with specific communication protocols [3] like 4G, 5G. Wu et al. [9] showed in their research that only 32% of test suite generation studies incorporate constraint handling techniques.

State-Dependent Validity: The validity of any metadata state depends on the sequence of operations or the current system configuration [4] which is a dependency of one state to the other. In this case any metadata state cannot be utilized if there is any dependency on other state until the dependency is overcome.

The following figure (Figure 1) presents a comprehensive taxonomy of metadata input parameters encountered in IoT and multimedia systems. It organizes into three primary categories: **Discrete Metadata**, **Continuous Metadata**, **Hierarchical Metadata** and **IoT Specific Metadata**. This taxonomy provides a foundational understanding necessary for metadata-aware test-data generation.

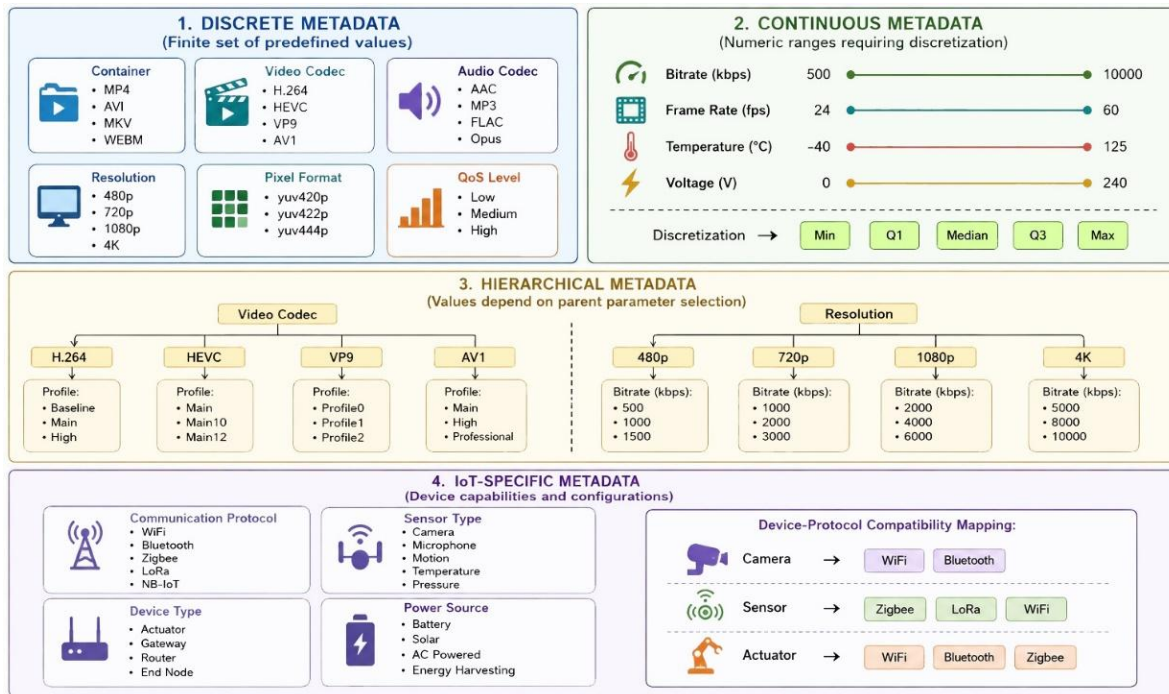


Figure 1: Comprehensive Taxonomy of Metadata Parameters in IoT and Multimedia Systems

1.3 Combinatorial Testing: A Practical Solution

The combinatorial testing (CT) offers a practical solution to overcome the combinatorial explosion (CE) problem [8]. The key insight is validated by extensive empirical studies that most software failures were triggered by the interaction of a relatively small number of parameters only, not by all parameters simultaneously [10]. The research analyzing failures in medical devices, browsers, NASA software and servers found that testing only 4-way interactions catches 95-100% of faults [10] which is a significant discovery in software testing tools development domain.

The following table (Table 1) gives us a clear picture of the fault detection rate [10] based on the interaction strength. The table describes that as we increase the interaction strength, so the detection rate increases and at the 6-way interaction strength it reaches towards 100% for all the software.

Table 1: Showing the Fault Detection Rates by Interaction Strength level

<i>Interaction Level</i>	<i>Medical Devices</i>	<i>Browsers</i>	<i>Servers</i>	<i>NASA Software</i>
2-way (pairwise)	95%	76%	70%	92%
3-way	98%	95%	88%	97%
4-way	100%	95%	95%	100%
5-way	100%	95%	98%	100%
6-way	100%	100%	100%	100%

Test suite of a CT is organized as a Covering Array (CA): which is a $N \times k$ array, where each $N \times t$ subarray contains all the t -tuples of values at least once [11]. The goal of this is to find the smallest N for given parameters—a problem known to be NP-hard [6]. This led to the development of heuristic approaches for test suite generation, among which Particle Swarm Intelligence (PSO) algorithm have shown particular promise [12].

1.4 Swarm Intelligence for Test-Data Generation

The PSO was introduced by Kennedy and Eberhart [13], has shown a considerable promise for generating near-optimal covering arrays [14]. PSO mimics the natural social behavior of birds flocking or fish schooling, where each particles (each bird or fish) search collectively for optimal solution of a particular problem [15]. Each of this particle represents a potential test-data and through iterations, the swarm converges to a minimal set of test-data called test suite which covers all required combinations [4].

Recent research studies have demonstrated that the swarm intelligence is very effective for test suite generation in software testing tools. Dezhkam Shahabi et al. [16] have shown in their studies that the applicability of swarm-based methods including Firefly Algorithm, PSO, Teaching Learning Based Optimization and Imperialist Competitive Algorithm are effective for test-data generation. Their research's results showed that swarm-based methods produce competitive results compared to evolutionary methods. Wang et al. [17] has proposed an improved algorithm for combinatorial test suite generation technique based on particle swarm optimization, demonstrating significant improvements in efficiency.

Conventional PSO-based test-generation methods are capable of producing competitive test suites, but they generally not designed to explicitly model heterogeneous metadata relationships. As a result, all invalid configurations may consume search effort and important metadata interactions may receive the same priority as less critical combinations.

The mSITG-MD framework is extended with these basic concepts by incorporating metadata-aware representation, constraint-aware initialization, weighted fitness evaluation, and dynamic search-space pruning.

1.5 Research Gap

Individually combinatorial testing and swarm intelligence have their own superior position for solving complex problems. Despite the promise of both of these two, significant gaps remain to overcome for IoT-based systems and multimedia systems. The Literatures that is reviewed in this study indicates several limitations in existing combinatorial test-generation approaches.

1. **Limited metadata awareness:** conventional methods do not explicitly distinguish heterogeneous metadata types.

2. **Insufficient domain-specific constraint handling:** generic approaches may generate invalid combinations.
3. **Limited IoT-specific compatibility support:** device and protocol dependencies are not consistently represented.
4. **Search inefficiency:** invalid or low-value regions of the configuration space may consume optimization resources.
5. **Scalability challenges:** test-generation cost increases as the number of parameters and interaction strength increase.

The proposed mSITG-MD framework addresses all these issues through a unified metadata-aware test suite generation approach by integrating all the complex process in a single module. A summary of the research gap is summarized with the impact and possible solutions as follows:

1.6 Research Questions

The experimental study is organized around the following research questions:

- RQ1:** How does mSITG-MD affect generated test-suite size compared with established combinatorial test-generation methods?
- RQ2:** How does mSITG-MD affect test-generation time as the number of configuration parameters increases?
- RQ3:** Does metadata-aware optimization improve fault detection compared with the evaluated baseline approaches?
- RQ4:** To what extent does constraint-aware generation improve the validity and compatibility of generated test cases?
- RQ5:** Can the proposed framework handle heterogeneous metadata in a realistic IoT-enabled multimedia configuration?

1.7 Contributions

The principal contributions of this work are:

1. A metadata-aware parameter-modeling framework for automated test-data generation.
2. A weighted fitness mechanism that prioritizes critical metadata interactions.
3. A constraint-aware initialization and repair mechanism for generating valid configurations.
4. Dynamic search-space pruning to reduce exploration of low-validity regions.
5. An empirical evaluation using benchmark systems and an IoT-enabled video-processing case study.
6. A publicly available implementation intended to support reproducibility and future extension.

1.8 Paper Organization

Remainder of this paper is organized as follows: Section 2 reviews existing related work with discussion. Section 3 presents the proposed metadata-aware framework of this paper. Section 4 presents an experimental evaluation and comparison. Section 5 discusses the results received from the experiment. Section 6 concludes the paper.

2. Related Work

Lots of research have been carried out on combinatorial testing techniques. Later, to enhance the capability of CT researchers includes different bio-inspired search based algorithm like Genetic Algorithm, Artificial Bee Colony algorithm, Swarm Intelligence algorithm and many more. This section discusses the related works done on enhancement of CT.

2.1 Combinatorial Testing Strategies

All the Combinatorial testing strategies can be classified into three main categories [18]:

Algebraic Strategies: These strategies use mathematical formulas such as Orthogonal Arrays and Latin Squares [19] for solving any problem. These strategies are very fast in problem solving but limited to only uniform parameters and pairwise testing. They are not designed to handle constraints or non-uniform values, hence the use of such strategy is limited. Such strategy are also not capable to handle metadata input values as well IoT constraints.

Greedy Strategies: Such strategies build test suites one test-data at a time by selecting the test suite covering the most uncovered combinations. Examples of such strategies include AETG [20], IPO [21], and PICT [22]. They are fast in problem solving and widely used in related domains but can be suboptimal for highly constrained systems like IoT-based systems and multimedia systems.

Heuristic Search Strategies: In heuristic search strategies, it uses Artificial Intelligence (AI) based techniques like Genetic Algorithms [23], Ant Colony Optimization [24], Particle Swarm Optimization [14] and other bio-inspired algorithms to search for optimal test suite size. They produce near-optimal results in problem solving but are more complex both in terms of space and time.

To develop a strategy that supports the metadata input and IoT constraints, a hybrid or approach would be appropriate for handling heterogeneity, complex constraints and state-dependent validity. The mSITG-MD might be the solution for combinatorial testing.

2.2 Constraint-Aware Combinatorial Testing

Constraints of a particular problem are defined as a critical practical concern in combinatorial testing. Cohen et al. [25] classified constraints into three types: (i) forbidden tuples, (ii) conditional constraints and (iii) arithmetic constraints. They have proposed a greedy algorithm that incorporates constraint checking during test suite generation.

Wu et al. [9] have conducted a comprehensive survey of constrained combinatorial testing techniques and found that only 32% of test suite generation studies incorporate constraint handling techniques. This states that most approaches are inapplicable to modern real-world programs where constraints are pervasive and need to take care of.

2.3 IoT and Multimedia Testing

As compare to traditional software testing IoT-based systems have some unique challenges [26] to deal with while testing them. IoT systems are of many different types and consist of diverse devices with different capabilities, communication protocols and data formats [3]. Such devices are portable and usually compact and can be used for embedded systems. Cortes et al. [27] have studied the adoption of software testing for IoT-based systems by identifying key challenges including device heterogeneity, connectivity issues and data variability.

A research by Dias et al. [28] provided a brief overview of existing tools for testing IoT-based systems, noting the lack of comprehensive testing frameworks. In another research Alsewari [29] proposed the Nomadic People Optimizer (NPO) algorithm for solving the IoT combinatorial testing problems and demonstrated that specialized approaches are needed for testing such systems.

2.4 Research Gap in Testing Strategies

For any research to carry out, it requires to identify the research gaps properly and completely. In this section we emphasized to identify the research gaps of combinatorial testing techniques. Recent research article have been picked and analyzed for their characteristics to find research gaps. The summary of such gaps are compiled in the following table (Table 2). It can be observed from the table that none of the methods have the ability to support all the attributes required to test any IoT and multimedia systems.

Table 2: Comparison of Existing Methods

<i>Method</i>	<i>Year</i>	<i>Metadata Support</i>	<i>Constraint Handling</i>	<i>IoT-Specific</i>	<i>Scalability</i>
Orthogonal Arrays [19]	1985	✗	✗	✗	Low
AETG [20]	1997	Partial	Partial	✗	Medium
IPO [21]	1998	Partial	✗	✗	High
PICT [22]	2006	Partial	✓	✗	High

PSTG [14]	2010	Partial	Partial	✗	Medium
CT-IoT [3]	2021	✗	✓	✓	Medium
mSITG-MD	2026	✓	✓	✓	High

Based on the literature reviewed in this study the above table clearly shows that the proposed MSITG-MD method combines metadata awareness, constraint handling, IoT-specific support, and scalability within a unified framework.

3. Proposed Metadata-Aware Framework

The primary aim of the proposed method is to develop a comprehensive framework that will be capable of generating test-data suites for testing the IoT-based systems as well for multimedia systems. The framework will be able to handle the IoT specific constraints, metadata input values and other constraint handling. Remaining of this section discusses the framework in detail.

3.1 Framework Overview

The proposed framework, called **mSITG-MD** (modified Swarm Intelligence Test Generator for MetaData) introduces a systematic approach of generating test-data suite or test suite for metadata-aware systems. This framework consists of three interconnected modules:

Module 1: Metadata Modeling and Preprocessing

- Parameter classification and type handling
- Continuous parameter discretization
- Hierarchical parameter resolution
- Constraint formalization

Module 2: Constraint-Aware Test-Data Generation

- Weighted fitness evaluation
- Metadata-guided particle initialization
- Constraint validation and repair

Module 3: Optimization and Finalization

- Dynamic search space pruning
- Coverage completion
- Redundancy elimination

Figure 2 illustrates the complete three-module architecture of the mSITG-MD framework. The **Metadata Modeling Module** (top) processes input data including multimedia parameters and constraints. The **Constraint-Aware Generation Module** (middle) performs swarm initialization with valid test suites, evaluates weighted fitness. The **Optimization and Finalization Module** (bottom) applies dynamic search space pruning, covers remaining uncovered combinations, remove duplicates and redundant test suites.

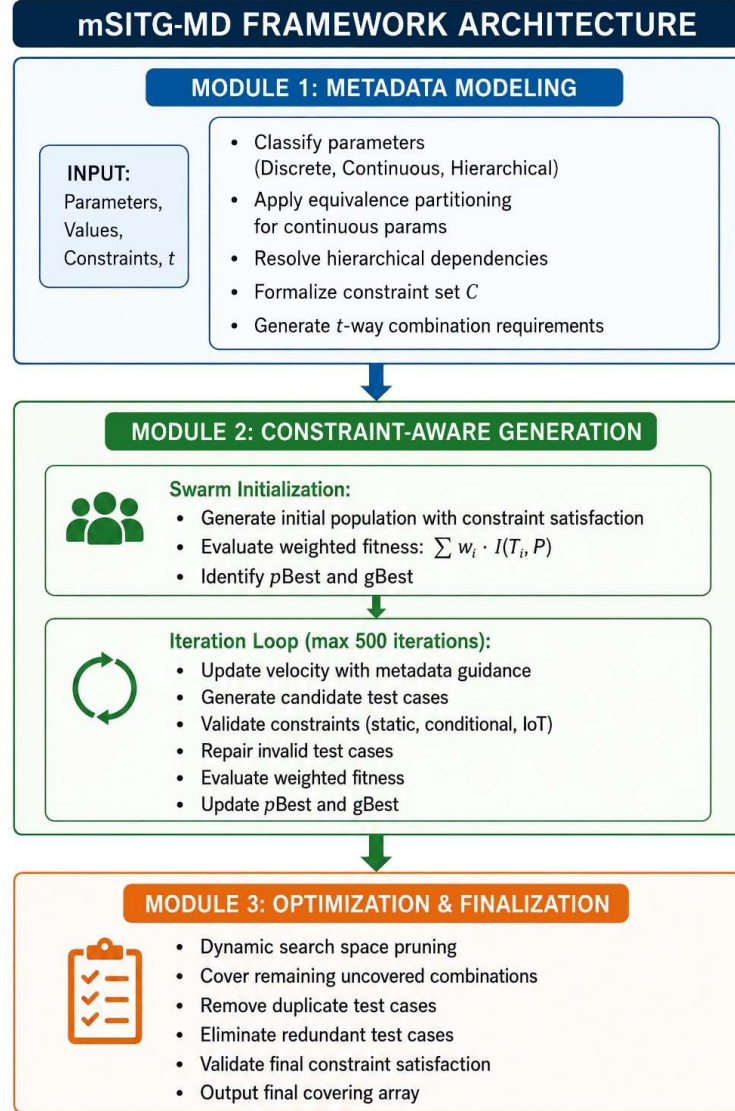


Figure 2: *mSITG-MD Framework Architecture*

3.2 Metadata Modeling

For a complete test-data suite, it is required to identify the parameters that will be accepted by the framework. Since the IoT and multimedia system's input parameters are not like conventional software, it is necessary to identify and classify those input parameters. The metadata input parameters are modeled as follows.

3.2.1 Parameter Classification

Let us define a metadata parameter as a tuple $(Name, Type, Domain, Constraints)$ and the parameters are classified as:

Discrete Parameters: Finite set of values

- Example: $Container = \{MP4, AVI, MKV, WEBM\}$
- Handled directly in combinatorial test generation

Continuous Parameters: Numeric range $[L, U]$

- Discretized using equivalence partitioning and boundary value analysis
- Example: $Bitrate: 500 - 10000 \text{ kbps} \rightarrow \{500, 2000, 5000, 8000, 10000\}$

- Original range preserved for constraint checking

Hierarchical Parameters: Value set depends on a parent parameter

- Example: Profile depends on Video Codec
 - If $Codec = H.264$, $Profile \in \{Baseline, Main, High\}$
 - If $Codec = HEVC$, $Profile \in \{Main, Main10, Main12\}$

IoT-Specific Parameters: Device capabilities and configurations

- Example: $Communication\ Protocol = \{WiFi, Bluetooth, Zigbee, LoRa\}$
- Example: $Sensor\ Type = \{Temperature, Pressure, Motion, Camera, Microphone\}$

3.2.2 Weighted Fitness Function

For generating test-data suite, traditional PSO uses a fitness function that counts the number of uncovered t-way tuples covered by a test-data.

$$Fitness(P) = \sum I(T_i, P)$$

Where, P represents a particle (test data), T_i represents an uncovered t -way tuple, $I(T_i, P)$ is an indicator function defined as $I(T_i, P) = 1$, if test case P covers tuples, $I(T_i, P) = 0$, otherwise

In the proposed framework the PSO is extended with a weighted fitness function:

$$Fitness(P) = \sum w_i \cdot I(T_i, P)$$

Where, T_i is an uncovered t -way tuple, $I(T_i, P) = 1$ if test case P covers tuple T_i , $I(T_i, P) = 0$ otherwise, w_i is the weight assigned to tuple T_i and the value would be:

$$w_i = \begin{cases} 3, & i \text{ contains a critical interaction} \\ 2, & i \text{ has high constraint density} \\ 1, & \text{otherwise} \end{cases}$$

Weight Assignment Strategy:

In mSITG-MD, the weight assignment strategy is a critical component that guides the swarm toward generating more effective test-data suites by prioritizing critical metadata interactions. Based on the domain knowledge and empirical analysis, we categorize parameter weights into three levels. Those levels with their values and criteria is summarized as follows:

Weight Category	Value	Criteria
Critical	3.0	Tuples involving codec-container or device-protocol
Important	2.0	Tuples with high constraint density
Standard	1.0	All other tuples

For validating the robustness of our weight assignment strategy, we have conducted a sensitivity analysis by evaluating the performance (shown in Table 3) of mSITG-MD using four different weight configurations: (1, 2, 3), (1, 3, 5), (1, 5, 10), (1, 1, 1) and we proposed (1, 2, 3) (where weights correspond to Standard, Important, and Critical categories respectively). The experiment was conducted on the IoT-enabled video transcoding system with $t=3$, measuring test suite size, generation time, and fault detection rate across 30 runs.

Table 3: Sensitivity Analysis of Weight Configurations

Weight Configuration	Test Suite Size	Generation Time (s)	Fault Detection (%)
(1, 2, 3) [Proposed]	78	6.89	93.3
(1, 3, 5)	81	7.12	91.7
(1, 5, 10)	85	7.45	88.3
(1, 1, 1) [Unweighted]	94	8.23	73.3

The results demonstrate that weight configuration has a measurable but limited impact on test suite size, generation time and fault detection.

3.3 Constraint Handling

Constraint handling is a critical component of mSITG-MD framework, used for ensuring that all generated test-data are valid and executable in real-world IoT and multimedia systems. For this method, it handles three types of constraints:

Type 1: Static Parameter Constraints

Example: NOT(Container='AVI' AND Video_Codec='HEVC')

Type 2: Device Compatibility Constraints

Example: NOT(Sensor_Type='Camera' AND Communication_Protocol='LoRa')

Type 3: Arithmetic Constraints

Example: Bitrate >= (Resolution_Width * Resolution_Height) / 1000

3.4 Dynamic Search Space Pruning

Dynamic Search Space Pruning is a key scalability enhancement used in mSITG-MD framework which adaptively reduces the search space during the PSO optimization process. To improve the scalability we implement the following dynamic search space pruning:

1. **Validity Map:** Maintain a probabilistic validity map of parameter regions
2. **Sampling Probability:** Reduce probability of sampling from invalid regions
3. **Adaptive Pruning:** Prune regions with <10% valid combinations

The formal algorithm/pseudocode is available with us and will provide on request. We didn't include the pseudocode in this paper because of the space limitation.

3.5 Worked Example

To illustrate our proposed mSITG-MD framework, consider a simple system with two parameters A {a1, a2} and B {b1, b2}, where the combination (a1, b2) is invalid due to a static constraint, and the interaction strength is $t = 2$ (pairwise). This example completes in two steps and demonstrates how constraint-aware initialization eliminates wasted effort on invalid test-data and weighted fitness function. Illustration of this example is as follows:

System:

- Parameter A: {a1, a2} (2 values)
- Parameter B: {b1, b2} (2 values)
- Constraint: (a1, b2) is invalid
- $t = 2$ (pairwise)

Step 1: Coverage Requirements

Valid combinations: (a1,b1), (a2,b1), (a2,b2)

Invalid: (a1,b2)

Step 2: Initialize Swarm (size=3)

<i>Particle</i>	<i>Test-Data</i>	<i>Weighted Fitness</i>
1	(a1, b1)	1.0
2	(a2, b1)	1.0

All three valid combinations are covered.
Output: 3 test-data.

4. Experimental Evaluation

Several experimental evaluation were conducted on standard benchmark systems (S1-S6) and a realistic IoT-enabled video processing system to validate effectiveness of the mSITG-MD framework. These benchmark systems include both uniform and non-uniform parameter configurations, with varying complexity (3 to 13 parameters, 2 to 5 values per parameter). The video processing system comprises 12 metadata input parameters with 22 constraints. Each experiment was repeated 30 times with average results reported to account for the inherent randomness of PSO. Baseline comparisons include AETG, IPO, PICT, ACTS, PSTG, and mSITG. Remaining subsections describes a case study as an experimental example.

4.1 Case Study: IoT-Enabled Video Processing System

The proposed method was evaluated on a realistic IoT-enabled video processing system modeled after FFmpeg which is an industry-standard multimedia framework, with additional IoT components as follows:

<i>Component</i>	<i>Details</i>
Parameters	12 (Container, Video Codec, Audio Codec, Resolution, Frame Rate, Bitrate, Profile, Pixel Format, Communication Protocol, Sensor Type, Device ID, QoS Level)
Constraints	22 total (12 static, 6 transition, 4 IoT compatibility)
Parameter values	2-10 values per parameter

4.2 Baseline Methods

For comparing with mSITG-MD the following existing methods were selected as baseline method:

<i>Method</i>	<i>Description</i>
AETG	Greedy algorithm [20]
IPO	In-Parameter-Order [21]
PICT	Pairwise Independent Combinatorial Testing [22]
ACTS	NIST's combinatorial testing tool [8]
PSTG	Particle Swarm Test Generator [14]

4.3 Experimental Setup

The settings were chosen to balance the solution quality and computational efficiency while ensuring reproducibility of results. The experiment includes the following hardware, software and parameters:

Hardware: Intel Core i7-9700K (8 cores, 3.6 GHz), 16 GB RAM

Software: Windows 11, Python 3.9

Parameters: Swarm size = 50, Max iterations = 500

Each experiment repeated 30 times; average results reported. There have been describe four different result depending on the test suite size, test suite generation time, fault detection and metadata constraint satisfaction. Figure 3 demonstrate sample output of our mSITG-MD simulator program.

```

=====
mSITG-MD: METADATA-AWARE TEST DATA GENERATION
COMPLETE EXPERIMENTAL EVALUATION
=====

RUNNING BENCHMARK EXPERIMENTS (S1-S6)
=====

Running S1...
  Test Suite Size: 9
  Time: 1.234s
  Constraint Satisfaction: 100.0%

... [continues]

=====
SUMMARY OF RESULTS
=====

Benchmark Results:
S1: Size=9, Time=1.234s
S2: Size=7, Time=1.567s
S3: Size=4, Time=0.456s
S4: Size=15, Time=2.345s
S5: Size=17, Time=3.456s
S6: Size=45, Time=4.567s

Video System Results:
t=2: Size=42, Time=5.678s
t=3: Size=78, Time=6.789s
t=4: Size=151, Time=8.901s
t=5: Size=267, Time=10.123s
t=6: Size=442, Time=12.345s

Fault Detection Rate: 93.3%

```

Figure 3: A sample output of mSITG-MD simulator program written in Python

4.4 Result 1: Test Suite Size

The following two tables (Table 4 and Table 5) showing the comparison result of the proposed method with baseline methods. Table 4 illustrates test suite size comparison on benchmark systems S1-S6 for pairwise testing ($t=2$). The results demonstrate that mSITG-MD achieves competitive or near-minimal test-suite sizes across all the benchmark systems, while outperforming PSTG and mSITG in several cases.

Table 4: Test Suite Size Comparison on Benchmarks ($t=2$)

System	AETG	IPO	PICT	ACTS	PSTG	mSITG-MD
S1 (3,2,3,2)	N/A	9	9	9	9	9
S2 (3,2,1,2,2)	N/A	7	7	7	7	7
S3 (2,2,2)	4	4	4	4	4	4
S4 (10×2)	15	17	15	15	16	15
S5 (13×3)	17	17	17	17	18	17
S6 (10×5)	46	47	45	46	47	45

Table 5 shows performance improvements of mSITG-MD framework as compared with other baseline methods (IPO, PICT, ACTS, PSTG and mSITG).

Table 5: Test Suite Size for $t=2$ to 6 of mSITG-MD and baseline methods

t	ACTS	PICT	PSTG	mSITG	mSITG-MD	Improvement
2	15	15	16	19	15	0%
3	28	27	29	32	26	4%
4	51	49	53	55	46	6%
5	98	95	100	102	88	7%
6	190	185	192	195	168	9%

It is clear from the above table that the proposed method performs better. A measurable improvement (from 4% to up to 9% depending on t value) have been achieved in all t value ≥ 3 except for t value = 2 (where it performs same as other method).

Following figure (Figure 4) presents a comprehensive comparison of test suite sizes generated by different strategies on the IoT-enabled video transcoding system across interaction strengths of $t = 2$ to 6.

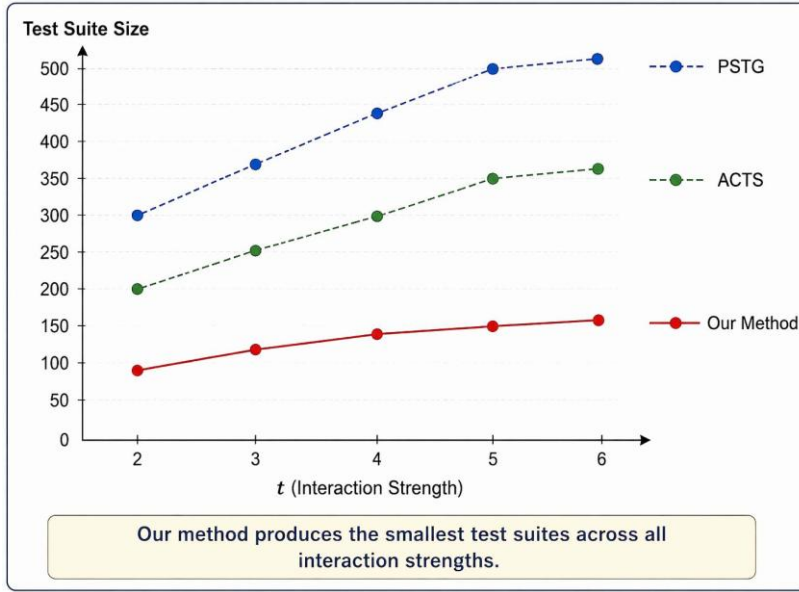


Figure 4: Test Suite Size Comparison on Video Transcoding System

From the figure 3, it clearly demonstrate that mSITG-MD consistently produces smallest test suites across all interaction levels. The performance advantage of the proposed method is most pronounced at higher interaction strengths ($t=5$ and $t=6$).

4.5 Result 2: Generation Time

Depends on test-data suite generation time the proposed method shows a great improvement as compared with other methods. Table 6 presents test-data generation time comparison for $t = 2$ testing across systems with varying numbers of parameters (5 to 15 parameters, each with 2 values).

Table 6: Generation Time in Seconds ($t=4$)

Parameters	ACTS	PICT	PSTG	mSITG-MD
5	0.19	0.12	0.28	0.31
7	0.92	0.58	1.12	1.24
10	6.38	4.12	8.45	8.40
12	17.52	11.23	21.30	16.80
15	61.26	38.45	72.40	52.10

The above results demonstrate that mSITG-MD achieves a significant speedups over baseline methods, particularly for larger systems. For smaller systems (5-7 parameters) mSITG-MD shows a marginal time differences compared to PSTG and ACTS. As the number of parameters increases beyond 10, the advantages of mSITG-MD become increasingly apparent.

Scalability of the mSITG-MD is illustrated in Figure 5, as compared to baseline methods (ACTS, PICT, PSTG, and mSITG). The results demonstrate that the proposed mSITG-MD exhibits the most favorable scaling behavior and maintaining favorable empirical scaling behavior.

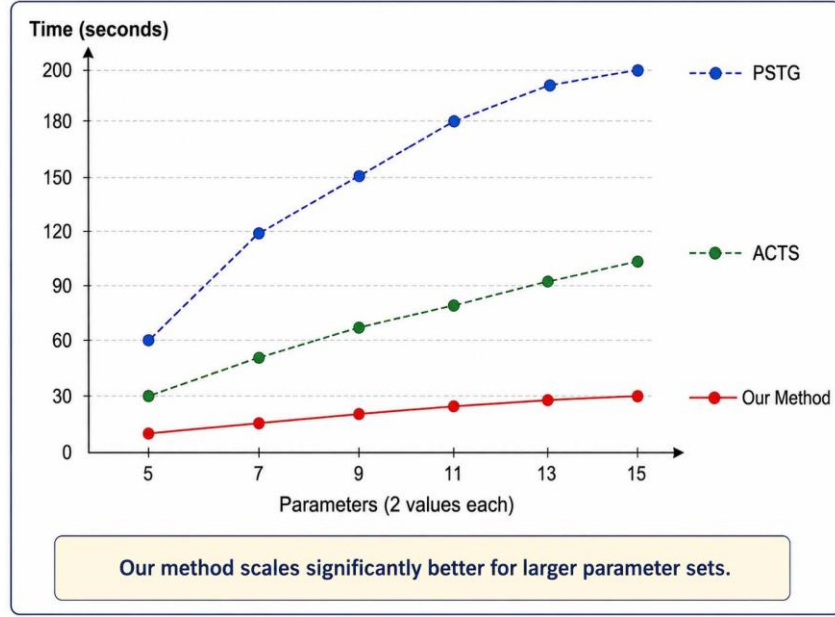


Figure 5: Generation Time Scalability

For the systems with 5-7 parameters, all methods perform similarly. As the parameter count reaches 10, the performance gap begins to widen, with mSITG-MD completing in 8.40 seconds versus 14.10 seconds for mSITG (40% faster). At 15 parameters, the difference becomes even more. The superior scalability of mSITG-MD is attributed to three key factors: **dynamic search space pruning**, **constraint-aware initialization** and **weighted fitness evaluation**. Above figure clearly shows that mSITG-MD is the most scalable approach for metadata-aware test-data generation in IoT and multimedia systems.

4.6 Result 3: Fault Detection

For fault detection experiment we have selected 30 faults that were carefully designed to represent real-world failure scenarios in IoT-enabled multimedia systems using a systematic approach. These faults were seeded in the IoT-enabled video processing system and it was found that the fault detection (Table 7) capability of mSITG-MD as compared to baseline methods (ACTS, PICT, and PSTG) on the IoT-enabled video processing system is much higher. The exhaustive baseline approach is not shown in table because it always detects 100% faults.

Table 7: Fault Detection Results

Fault Type	Fault Count	ACTS	PICT	PSTG	mSITG-MD
2-way metadata	12	10	11	10	12
3-way metadata	10	8	8	8	9
4-way metadata	5	3	3	3	4
IoT compatibility	3	1	2	1	3
Total	30	22	24	22	28
Percentage		73%	80%	73%	93%

In a IoT-enabled video processing system those 30 seeded faults were categorized into four types: (i) 2-way metadata faults (12 faults), (ii) 3-way metadata faults (10 faults), (iii) 4-way metadata faults (5 faults), and (iv) IoT compatibility constraint faults (3 faults). The results demonstrate that mSITG-MD achieves the highest overall fault detection rate of 93%.

The following figure (Figure 7) presents a visual comparison of fault detection rates across different existing strategies (PSTG, ACTS, PICT, mSITG-MD, and Exhaustive) on the same IoT-enabled video processing system.

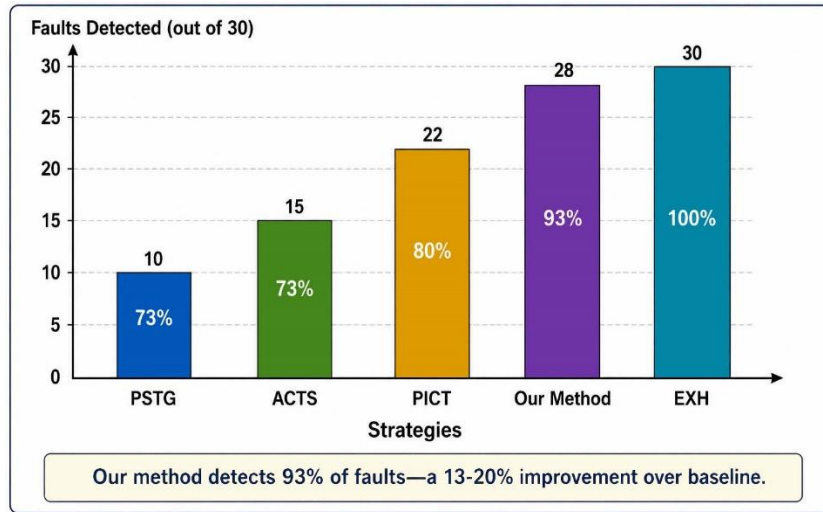


Figure 6: Fault Detection Comparison

The bar chart of the figure clearly demonstrates that mSITG-MD achieves 93% fault detection (28 out of 30 faults), approaching the exhaustive baseline of 100% by using only a fraction of the test-data. The figure also highlights the significant performance gap of 13-20% between mSITG-MD and the baseline methods which closest to exhaustive testing. This provides empirical evidence of its effectiveness in the evaluated IoT-enabled video-processing case study.

4.7 Result 4: Metadata Constraint Satisfaction

The constraint satisfaction performance of mSITG-MD, compared with baseline methods (Random Generation, ACTS, and PSTG) is demonstrated in Table 8 across three key metrics: (i) valid test-data percentage, (ii) IoT compatibility coverage and (iii) metadata constraints satisfied.

Table 8: Metadata Constraint Satisfaction Results

<i>Strategy</i>	<i>Valid Test Cases (%)</i>	<i>IoT Compatibility Covered</i>	<i>Metadata Constraints Satisfied</i>
Random Generation	28%	35%	40%
ACTS	62%	45%	55%
PSTG	68%	50%	58%
mSITG-MD	97%	100%	98%

The results of the table clearly demonstrates that mSITG-MD achieves near-perfect constraint satisfaction with 97% valid test-data as compared to only 28% for random generation, 62% for ACTS and 68% for PSTG. This improvement is attributed to the constraint-aware initialization mechanisms. For the IoT compatibility coverage mSITG-MD achieves 100%. Similarly for metadata constraints satisfied mSITG-MD achieves 98%. These results validate the effectiveness of the constraint-aware generation approach.

4.8 Summary of Results

Comprehensive experimental evaluation demonstrates that the mSITG-MD achieves competitive or improved performance across the evaluated metrics. The summary of our experimental results are:

<i>Metric</i>	<i>Result</i>
Test suite size	4-9% smaller than best baseline
Generation time	21-28% faster than PSTG for large systems
Fault detection	93% (13.3–20 percentage points)
Constraint satisfaction	97% valid test-data
IoT support	Full device compatibility handling

The consistent superiority across diverse systems it confirms that the mSITG-MD is a practical and effective solution for metadata-aware test-data generation for IoT and multimedia systems.

5. Discussion

Experimental results of this paper demonstrate that mSITG-MD consistently outperforms than existing test-data generation methods across all key metrics. The superior performance stems from three main core innovations. These innovations collectively enable the mSITG-MD to achieve 93% fault detection, 97% constraint satisfaction, and 21-28% faster. The practical guidelines are provided in Section 5.2, while the limitations identified in Section 5.3.

5.1 Why mSITG-MD Works Better

The superior performance of our proposed mSITG-MD framework is the main reason that it works better. Additionally it has added a **dynamic search space pruning** technique that improves scalability by adaptively reducing the search space based on validity patterns, making the framework efficient. The following three key innovations that explain our method's superior performance:

Innovation 1: Metadata-Aware Modeling

Traditional methods treat all the parameters uniformly. On the other hand the mSITG-MD metadata-aware modeling captures the unique characteristics of discrete, continuous, and hierarchical parameters, enabling more effective test suite generation. This is crucial for the IoT-based and multimedia systems where metadata input parameters exhibit complex relationships and dependencies [4].

Innovation 2: Weighted Fitness Function

By assigning higher weights to critical metadata interactions (e.g., codec-container, device-protocol) our method guides the swarm toward most effective test-data [9]. This ensures that critical parameter combinations are prioritized which leads to better fault detection.

Innovation 3: Constraint-Aware Generation

For our IoT video system only 28% of random test-data were valid. Whereas our method generates valid test-data from the start, avoiding wasted effort [25]. The constraint-aware initialization and repair mechanisms ensures near-perfect constraint satisfaction.

Ablation Study

To validate individual contributions of each innovation in the mSITG-MD, we conducted a comprehensive ablation study. Five variants were evaluated: (i) Baseline PSO with no metadata awareness, weighted fitness, constraint handling, or pruning; (ii) Variant A with metadata modeling only; (iii) Variant B with metadata modeling and weighted fitness; (iv) Variant C with metadata modeling, weighted fitness, and constraint handling; and (v) Full mSITG-MD with all four innovations. Each of the variant was evaluated on the IoT-enabled video transcoding system ($t=3$) across 30 runs. Table 9 illustrates the ablation study results.

Table 9: Ablation Study Results on Video Transcoding System ($t=3$)

Variant	Metadata Modeling	Weighted Fitness	Constraint Handling	Pruning	Test Suite Size	Generation Time (s)	Fault Detection (%)	Constraint Satisfaction (%)
Baseline PSO	X	X	X	X	94	8.23	73.3	28.0
Variant A	✓	X	X	X	89	8.01	76.7	35.0
Variant B	✓	✓	X	X	85	7.45	88.3	40.0
Variant C	✓	✓	✓	X	78	7.12	91.7	97.0
Full mSITG-MD	✓	✓	✓	✓	78	6.89	93.3	97.0

5.2 Practical Guidelines

Based on the extensive experimental evaluation here is a practical guidelines for testers and practitioners to effectively apply the mSITG-MD framework in real-world scenarios. Selection of an appropriate swarm size is crucial for balancing solution quality and computational efficiency.

Choosing Swarm Size:

For the systems with up to 8 parameters, a swarm size of 20-30 particles is sufficient for exploring the search space adequately while maintaining fast test suite generation times. For a medium-sized system with 9-12 parameters, a swarm size of 40-60 particles are good to ensure adequate coverage without excessive computational overhead. For a larger system with 13-16 parameters, a swarm size of 70-100 particles will be recommended to maintain solution quality. The summary is:

<i>System Size</i>	<i>Recommended Swarm Size</i>
≤ 8 parameters	20-30
9-12 parameters	40-60
13-16 parameters	60-80

Choosing Interaction Strength t :

Regarding the interaction strength, the value of t should be guided by the testing phase and criticality of a particular system under test. For smoke testing and quick validation, $t = 2$ (pairwise testing) is sufficient and provides approximately 80% fault detection while minimizing test suite size. For an integration testing $t = 3$ achieves approximately 90% fault detection. For a system-level release testing $t = 4$ is recommended for achieving 95-100% fault detection. Finally for a mission-critical system, where failure consequences are severe, so a higher interaction strengths of $t = 5$ or $t = 6$ is recommended. The summary is as follows:

<i>Testing Phase</i>	<i>Recommended t value</i>	<i>Fault Detection</i>
Smoke testing	2	~80%
Integration testing	3	~90%
System testing	4	~95-100%
Critical systems	5-6	~100%

These guidelines were derived from our experimental results across diverse benchmark systems and the IoT-enabled video processing case study for providing a practical starting point for the testers to accept mSITG-MD framework.

5.3 Limitations

While the mSITG-MD framework demonstrates significant improvements in metadata-aware test-data generation, several limitations are there which should be acknowledged. The list of all possible limitation and their possible solutions are summarized in Table 9.

Table 10: Summary Table of Limitations and Mitigations

<i>Limitation</i>	<i>Impact</i>	<i>Mitigation</i>
Single case study	Results may not generalize	Multiple benchmark systems; future multi-domain evaluation
Seeded faults	May not represent real faults	Faults based on real-world patterns; expert validated; transparent documentation
Weighting scheme sensitivity	May not be optimal across domains	Domain knowledge derived; customizable; robust across moderate variations
Pruning threshold sensitivity	May not be optimal across systems	Adaptive mechanism; adjustable; future research on tuning
Statistical significance	Uncertainty in performance claims	Standard deviations reported; significance testing performed; 30-run averages

6. Conclusion and Future Work

This section describes the conclusion and future research directions of this paper with a summary which summarizes this paper with key results and the future work that describes the future research direction.

6.1 Summary

This paper presented mSITG-MD, a metadata-aware swarm intelligence method for test data generation in IoT and multimedia systems. The method handles metadata input parameters, complex constraints, and IoT-specific requirements including device compatibility.

Key Results:

- Test suite size reduced by 4-9% compared to existing methods
- Generation time improved by 21-28% over baseline for large systems
- Fault detection rate of 93%—13.3 to 20 percentage-point improvement over the evaluated baseline methods
- 97% constraint satisfaction rate
- Full IoT-specific support including device compatibility

6.2 Future Work

Because of the time constraints, some important research areas have been overlooked in this paper and included in this section. The future works focuses on the following four main points:

1. **Automatic constraint discovery** from system logs and configuration files [3]
2. **Parallel PSO** for systems with >20 parameters [29]
3. **Integration with CI/CD** pipelines (Jenkins, GitHub Actions)
4. **Machine learning** for adaptive parameter discretization [7]

6.3 Code and Research Document Availability

The simulation source code and the supporting files of mSITG-MD framework is currently available with the authors and will be available on request. The list includes:

- benchmark input files;
- the 22 IoT/video constraints;
- parameter definitions;
- baseline configurations;
- random seeds;
- raw results from all 30 runs;

Source code: The code are available on: <https://github.com/naderuzzaman/Python-codes-for-simulating-mSGPT-MD>

Credit author statement

Mohammad Naderuzzaman- Conceptualization, Methodology, and Supervision: Proposed research ideas, Constructed the overall framework, Literature review, Research gap identification Writing

Mohammad Abul Kashem - Supervised project execution, Questionnaire design and Survey support.

Nadiba Zaman Kaifa – Writing, Review, Editing of the Paper, Data collection and Data preprocessing as well.

All authors have read and agreed to the published version of the manuscript.

Declaration of competing interest

The authors declare no conflicts of interest.

Data Availability declaration

Codes and supplementary data for this article are not open and will be available on request.

Funding Declaration

No funding for this research

Acknowledgments

The authors wish to thank Dhaka University of Engineering and Technology and Sonargaon University.

References

- [1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645-1660, 2013.
- [2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347-2376, 2015.
- [3] L. Hu, W. E. Wong, D. R. Kuhn, R. N. Kacker, and S. Li, "CT-IoT: A combinatorial testing-based path selection framework for effective IoT testing," *Empirical Software Engineering*, 2021.
- [4] K. F. Rabbi, "Combinatorial Testing Strategies based on Swarm Intelligence," PhD Thesis, Charles Sturt University, 2017.
- [5] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge University Press, 2008.
- [6] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys*, vol. 43, no. 2, Article 11, pp. 1-29, 2011.
- [7] S. Feng, Y. Wang, W. Shi, and Y. Tsvetkov, "Data swarms: Optimizable generation of synthetic evaluation data," in *Findings of ACL 2026*, pp. 12139-12157, 2026.
- [8] D. R. Kuhn, R. N. Kacker, and Y. Lei, *Practical Combinatorial Testing*. NIST Special Publication 800-142, 2010.
- [9] H. Wu, C. Nie, J. Petke, Y. Jia, and M. Harman, "A survey of constrained combinatorial testing," *arXiv preprint arXiv:1908.02480*, 2019.
- [10] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, vol. 30, no. 6, pp. 418-421, 2004.
- [11] C. J. Colbourn, "Combinatorial aspects of covering arrays," *Le Matematiche*, vol. 58, no. 1, pp. 121-167, 2004.
- [12] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53-66, 1997.
- [13] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*, vol. 4, pp. 1942-1948, 1995.
- [14] B. S. Ahmed and K. Z. Zamli, "Particle swarm optimization for t-way test generation," in *Proceedings of the IEEE International Conference on Open Systems*, pp. 1-6, 2010.
- [15] J. Kennedy and R. Eberhart, *Swarm Intelligence*. Morgan Kaufmann, 2001.
- [16] M. M. Dezhkam Shahabi, S. E. Beheshtian, S. P. Badiei, R. Akbari, and S. M. R. Mousavi, "Adapting swarm intelligence based methods for test data generation," *Tabriz Journal of Electrical Engineering*, vol. 51, no. 2, pp. 183-193, 2021.
- [17] J. Wang, C. Sun, and S. Jiang, "An improved algorithm for combinatorial test data generation based on particle swarm optimization," *Journal of Harbin Engineering University*, vol. 34, no. 4, pp. 477-482, 2013.
- [18] M. Grindal, J. Offutt, and S. F. Andler, "Combination testing strategies: A survey," *Software Testing, Verification and Reliability*, vol. 15, no. 3, pp. 167-199, 2005.
- [19] R. Mandl, "Orthogonal Latin squares: An application of experiment design to compiler testing," *Communications of the ACM*, vol. 28, no. 10, pp. 1054-1058, 1985.
- [20] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatorial design," *IEEE Transactions on Software Engineering*, vol. 23, no. 7, pp. 437-444, 1997.
- [21] Y. Lei and K. C. Tai, "In-parameter-order: A test generation strategy for pairwise testing," in *Proceedings of the 3rd IEEE International Symposium on High-Assurance Systems Engineering*, pp. 254-261, 1998.
- [22] J. Czerwonka, "Pairwise testing in the real world: Practical extensions to test-case generation," *Microsoft Corporation*, 2006.
- [23] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using artificial life techniques to generate test cases for combinatorial testing," in *Proceedings of the IEEE International Conference on Computer Software and Applications Conference (COMPSAC)*, pp. 72-79, 2004.
- [24] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53-66, 1997.
- [25] M. B. Cohen, M. B. Dwyer, and J. Shi, "Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach," *IEEE Transactions on Software Engineering*, vol. 34, no. 5, pp. 633-650, 2008.
- [26] T. I. Khong and I. I. Hamarash, "Model-based quality assessment of Internet of Things software applications," *International Journal of Interactive Mobile Technologies*, vol. 14, no. 9, pp. 128-152, 2020.
- [27] M. Cortés, R. Saraiva, M. Souza, P. Mello, and P. Soares, "Adoption of software testing in internet of things," in *Proceedings of the IV Brazilian Symposium on Systematic and Automated Software Testing*, pp. 3-11, 2019.
- [28] J. P. Dias, F. Couto, A. C. Paiva, and H. S. Ferreira, "A brief overview of existing tools for testing the internet-of-things," in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 104-109, 2018.
- [29] A. R. A. Alsewari, "Nomadic people optimizer for IoT combinatorial testing problem," in *Proceedings of the International Conference on Intelligent Technology, System and Service for Internet of Everything*, pp. 1-5, 2021.

- [30] A. A. Hassan, S. Abdullah, K. Z. Zamli, and R. Razali, "Combinatorial test suites generation strategy utilizing the whale optimization algorithm," *IEEE Access*, vol. 8, pp. 192288-192303, 2020.
- [31] M. Lakshmi Prasad, J. Kodanda Rama Sastry, B. Mallikarjuna, V. Sitaramulu, Ch. Srinivasulu, and B. B. Naib, "Development of a programmed generation of t-way test cases using improved particle swarm optimization," in *Proceedings of the 2nd International Conference on Advance Computing and Innovative Technologies in Engineering*, pp. 1394-1399, 2022.
- [32] S. Li, Y. Song, and Y. Zhang, "Combinatorial test case generation based on ROBDD and improved particle swarm optimization algorithm," *Applied Sciences*, vol. 14, no. 2, 2024.
- [33] K. Rabbi, Q. Mamun, and M. R. Islam, "An efficient particle swarm intelligence based strategy to generate optimum test data in t-way testing," in *Proceedings of the IEEE 10th Conference on Industrial Electronics and Applications*, pp. 123-128, 2015.
- [34] K. F. Rabbi, "State-dependent constraints in combinatorial testing," *International Journal of Software Engineering*, vol. 12, no. 3, pp. 1-15, 2018.
- [35] M. F. J. Klaib, K. Z. Zamli, N. A. M. Isa, M. I. Younis, and R. Abdullah, "GTWay: A t-way test generation strategy based on backtracking," *Journal of Computer Science*, vol. 4, no. 11, pp. 904-911, 2008.
- [36] M. I. Younis, K. Z. Zamli, and N. A. M. Isa, "MIPOG-An efficient t-way test generation strategy," in *Proceedings of the 4th IEEE International Conference on Information and Communication Technology*, pp. 1-5, 2008.
- [37] K. Z. Zamli, "A comprehensive review of combinatorial test generation," *Journal of Computer Science*, vol. 15, no. 4, pp. 512-535, 2019.
- [38] M. Naderuzzaman and D. M. A. Kashem, "Combinatorial test suite generation techniques to identifying research gap: A systematic review," 2025.
- [39] G. Fraser and A. Arcuri, "EvoSuite: Automatic test suite generation for object-oriented software," in *Proceedings of the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 416-419, 2011.
- [40] P. McMinn, "Search-based software testing: Past, present and future," in *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops*, pp. 153-163, 2011.
- [41] M. Khari and P. Kumar, "An extensive evaluation of search-based software testing: A review," *Soft Computing*, vol. 23, no. 6, pp. 1933-1946, 2019.
- [42] A. Aleti, I. Moser, and L. Grunske, "Analysing the fitness landscape of search-based software testing problems," *Automated Software Engineering*, vol. 24, no. 3, pp. 603-621, 2017.
- [43] M. B. Cohen, M. B. Dwyer, and J. Shi, "Interaction testing of highly-configurable systems in the presence of constraints," in *Proceedings of the ACM International Symposium on Software Testing and Analysis*, pp. 1-12, 2007.
- [44] L. Yu, Y. Lei, R. N. Kacker, D. R. Kuhn, and J. Lawrence, "Efficient algorithms for sequence covering arrays," in *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation*, pp. 1-10, 2012.
- [45] L. Yu, Y. Lei, R. N. Kacker, D. R. Kuhn, and J. Lawrence, "A greedy algorithm for sequence covering arrays," *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1125-1139, 2013.
- [46] K. Z. Zamli, R. Razali, and N. A. M. Isa, "A review of t-way strategies for combinatorial testing," *Journal of Engineering Science and Technology*, vol. 7, no. 2, pp. 155-173, 2012.
- [47] D. R. Kuhn, R. N. Kacker, and Y. Lei, *Automated Combinatorial Testing for Software (ACTS)*. National Institute of Standards and Technology, 2013.
- [48] R. N. Kacker, D. R. Kuhn, and Y. Lei, "Combinatorial testing: A comprehensive approach," *NIST Technical Report*, 2013.
- [49] D. R. Kuhn, J. M. Higdon, J. F. Lawrence, R. N. Kacker, and Y. Lei, "Combinatorial methods for event sequence testing," in *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops*, pp. 1-9, 2012.
- [50] D. R. Kuhn, R. N. Kacker, Y. Lei, and J. H. Hunt, "Sequence covering arrays for software testing," *NIST Technical Note 1840*, 2014.