SCHEDULING ENGINE - TESTING SYSTEM

THEORETICAL FOUNDATION & MATHEMATICAL FRAMEWORK

TEAM - LUMEN [TEAM-ID: 93912]

October 4, 2025

Abstract

This document presents a mathematically rigorous, deterministic framework for generating comprehensive test cases for the 7-stage scheduling engine. The framework ensures complete validation coverage across all input CSV files, internal processing stages, and output generation, providing both synthetic data generation and expected output verification mechanisms. This system enables exhaustive unit testing, integration validation, and algorithmic fine-tuning through systematic, reproducible test case creation.

Contents

1 THEORETICAL FOUNDATION

1.1 Mathematical Framework

Let $\mathcal{T} = (I, P, O, V)$ be our test case generation system where:

- $I = \{i_1, i_2, \ldots, i_{12}\}$ represents the 12 input CSV files
- $P = \{p_1, p_2, \ldots, p_7\}$ represents the 7 processing stages
- $O = \{o_1, o_2, \ldots, o_k\}$ represents all output files and tables
- $V : I \times P \to O$ represents the validation function

1.2 Deterministic Generation Principle

For any test case $t \in \mathcal{T}$, we define the deterministic generation function:

$$G(t) = \prod_{i=1}^{12} G_i(seed, constraints) \to (CSV_i, Expected_{stages}, Expected_{output})$$

Where each G_i generates syntactically and semantically valid data following the relational constraints defined in our PostgreSQL schema.

2 INPUT DATA GENERATION FRAMEWORK

2.1 Core Institution Data Generation

2.1.1 Institutions.csv Generation

$$\mathcal{I}_{inst} = \{(id, tenant\_id, name, code, type, state, district)\}$$

Generation algorithm:

$$\begin{aligned}
institution_id &= UUID.v4() && (1)\\
tenant_id &= UUID.v4() && (2)\\
institution_name &= f"Test_Institution_\{seed\}" && (3)\\
institution_code &= f"INST\{seed:04d\}" && (4)\\
institution_type &\in \{GOVERNMENT, PRIVATE, AUTONOMOUS\} && (5)\\
state &= "Jharkhand" && (6)\\
district &\in \{Ranchi, Dhanbad, Bokaro, Jamshedpur\} && (7)
\end{aligned}$$

2.1.2 Departments.csv Generation

$$\mathcal{D}_{dept} = \{(dept\_id, tenant\_id, inst\_id, code, name, head\_faculty)\}$$

For each institution i, generate $n_d \in [3, 8]$ departments:

$$\begin{aligned}
department_count &= random(3, 8) && (8)\\
\forall j \in [1, department_count] &: && (9)\\
dept_id &= UUID.v4() && (10)\\
dept_code &= f"DEPT\{j:03d\}" && (11)\\
dept_name &\in \{Computer_Science, Electronics, Mechanical, Civil, Mathematics\} && (12)
\end{aligned}$$

2.1.3 Programs.csv Generation

$$\mathcal{P}_{prog} = \{(prog\_id, tenant\_id, inst\_id, dept\_id, code, name, type, duration, credits)\}$$

For each department d, generate $n_p \in [2, 4]$ programs:

$$program\_count = random(2, 4) \tag{13}$$

$$program\_type \in \{UNDERGRADUATE, POSTGRADUATE, DIPLOMA\} \tag{14}$$

$$duration\_years = \begin{cases} 3 & \text{if DIPLOMA} \\ 4 & \text{if UNDERGRADUATE} \\ 2 & \text{if POSTGRADUATE} \end{cases} \tag{15}$$

$$total\_credits = duration\_years \times 30 \tag{16}$$

2.1.4 Courses.csv Generation

$$\mathcal{C}_{course} = \{(course\_id, tenant\_id, inst\_id, prog\_id, code, name, type, theory\_hrs, practical\_hrs, credits)\}$$

For each program p, generate courses following curriculum distribution:

$$core\_courses = 0.6 \times total\_credits/3 \tag{17}$$

$$elective\_courses = 0.25 \times total\_credits/3 \tag{18}$$

$$skill\_courses = 0.15 \times total\_credits/3 \tag{19}$$

$$credits \in \{1, 2, 3, 4\} \tag{20}$$

$$theory\_hours = credits \times 15 \tag{21}$$

$$practical\_hours = \begin{cases} 0 & \text{if theory course} \\ credits \times 30 & \text{if lab course} \end{cases} \tag{22}$$

2.2 Operational Resources Generation

2.2.1 Shifts.csv Generation

$$\mathcal{S}_{shift} = \{(shift\_id, tenant\_id, inst\_id, code, name, type)\}$$

Standard shift generation:

$$shifts = \{MORNING, AFTERNOON, EVENING\} \tag{23}$$

$$\forall s \in shifts: \tag{24}$$

$$shift\_id = UUID.v4() \tag{25}$$

$$shift\_code = s[0:3] + \text{"}\_SHIFT\text{"} \tag{26}$$

$$shift\_type = s \tag{27}$$

2.2.2 Time_slots.csv Generation

$$\mathcal{T}_{slot} = \{(slot\_id, tenant\_id, inst\_id, shift\_id, code, day\_num, start\_time, end\_time)\}$$

For each shift s and each day $d \in [1, 6]$:

$$time\_slots\_per\_day = \begin{cases} 8 & \text{if MORNING shift} \\ 6 & \text{if AFTERNOON shift} \\ 4 & \text{if EVENING shift} \end{cases} \tag{28}$$

$$slot\_duration = 60 \text{ minutes} \tag{29}$$

$$start\_time\_base = \begin{cases} 09:00 & \text{if MORNING} \\ 14:00 & \text{if AFTERNOON} \\ 18:00 & \text{if EVENING} \end{cases} \tag{30}$$

2.2.3 Faculty.csv Generation

$\mathcal{F}_{faculty} = \{(faculty\_id, tenant\_id, inst\_id, dept\_id, code, name, designation, employment\_type, max\_hours, pre$

For each department d, generate faculty with distribution:

$$faculty\_per\_dept = random(5, 12) \tag{31}$$

$$designation\_distribution = \{PROFESSOR : 0.2, ASSOCIATE : 0.3, ASSISTANT : 0.4, LECTURER : 0.1 \tag{32}$$

$$max\_hours\_per\_week = random(12, 22) \tag{33}$$

$$preferred\_shift = random\_choice(available\_shifts) \tag{34}$$

2.2.4 Rooms.csv Generation

$\mathcal{R}_{room} = \{(room\_id, tenant\_id, inst\_id, code, name, type, capacity, dept\_relation, assigned\_depts)\}$

Room generation with capacity constraints:

$$total\_rooms = \lceil 1.5 \times total\_courses \rceil \tag{35}$$

$$room\_type\_distribution = \{CLASSROOM : 0.7, LABORATORY : 0.25, AUDITORIUM : 0.05\} \tag{36}$$

$$capacity = \begin{cases} random(30, 60) & \text{if CLASSROOM} \\ random(20, 30) & \text{if LABORATORY} \\ random(100, 300) & \text{if AUDITORIUM} \end{cases} \tag{37}$$

2.2.5 Equipment.csv Generation

$$\mathcal{E}_{equip} = \{(equip\_id, tenant\_id, inst\_id, code, name, type, room\_id, dept\_id, criticality)\}$$

Equipment generation for laboratory rooms:

$$\forall r \in rooms\_laboratory : \tag{38}$$

$$equipment\_count = random(3, 8) \tag{39}$$

$$criticality = \{CRITICAL : 0.4, OPTIONAL : 0.6\} \tag{40}$$

$$equipment\_types = \{Computer, Projector, Whiteboard, Lab\_Bench, Microscope\} \tag{41}$$

2.2.6 Student_data.csv Generation

$\mathcal{S}_{student} = \{(student\_id, tenant\_id, inst\_id, uuid, enrolled\_courses, preferred\_shift, academic\_year)\}$

Student generation with enrollment patterns:

$$students\_per\_program = random(80, 200) \tag{42}$$
$$enrolled\_courses\_count = random(4, 8) \tag{43}$$
$$course\_selection = core\_courses \cup random\_subset(elective\_courses, 2) \tag{44}$$
$$preferred\_shift = weighted\_random(shifts, [0.6, 0.3, 0.1]) \tag{45}$$

2.3 Relationship Mappings Generation

2.3.1 Faculty_course_competency.csv Generation

$\mathcal{FC}_{comp} = \{(comp\_id, faculty\_id, course\_id, competency\_level, preference\_score, years\_exp, cert\_status)\}$

Competency assignment with realistic constraints:

$$\forall f \in faculty, \forall c \in courses\_same\_dept : \tag{46}$$
$$assignment\_probability = \begin{cases} 0.8 & \text{if course matches faculty specialization} \\ 0.3 & \text{if course is general/basic} \\ 0.1 & \text{if course is advanced/specialized} \end{cases} \tag{47}$$
$$competency\_level = random(5, 10) \text{ if assigned} \tag{48}$$
$$preference\_score = competency\_level + random(-2, 2) \tag{49}$$
$$years\_experience = faculty\_age - 25 \tag{50}$$

2.3.2 Constraints.csv Generation

$$\mathcal{CN}_{const} = \{(const\_id, tenant\_id, code, name, type, expression, weight)\}$$

Standard scheduling constraints:

$$hard\_constraints = \{no\_overlap, faculty\_hours, room\_capacity\} \tag{51}$$
$$soft\_constraints = \{preferred\_times, room\_preferences, workload\_balance\} \tag{52}$$
$$constraint\_weight = \begin{cases} \infty & \text{if HARD constraint} \\ random(0.1, 1.0) & \text{if SOFT constraint} \end{cases} \tag{53}$$

3 INTERNAL PROCESSING VALIDATION

3.1 Stage-2: Student Batching Validation

Expected batch generation algorithm:

$$B_{expected} = BatchingAlgorithm(S_{students}, C_{courses}, R_{rooms})$$

Where batching follows multi-objective optimization:

$$\begin{aligned} & minimize && \sum_{b \in batches} |B_b| - optimal\_batch\_size|^2 && (54) \\ & subject\ to && \bigcup_b B_b = S_{students} && (55) \\ & && B_i \cap B_j = \emptyset \quad \forall i \neq j && (56) \\ & && |B_b| \leq max\_room\_capacity \quad \forall b && (57) \end{aligned}$$

Expected outputs:

- student_batches table: batch_count $\in [\lceil |students|/60 \rceil, \lceil |students|/30 \rceil]$
- batch_course_enrollment table: entries = $\sum_b |courses\_for\_batch\_b|$

3.2 Stage-3: Data Compilation Validation

Expected normalization results:

$$\mathcal{N} = Normalize(Arrays \to Relations)$$

$$course\_prerequisites = \{(c_i, c_j) | c_j \in prerequisites\_array[c_i]\} \tag{58}$$

$$room\_department\_access = \{(r_i, d_j) | d_j \in assigned\_departments[r_i]\} \tag{59}$$

Expected table sizes:

- course_prerequisites: $\leq 0.3 \times |courses|^2$
- room_department_access: $\leq |rooms| \times 2$

3.3 Stage-4: Feasibility Check Validation

Seven-layer feasibility validation:

$$F = (F_1 \wedge F_2 \wedge F_3 \wedge F_4 \wedge F_5 \wedge F_6 \wedge F_7)$$

Where:

$$F_1 = DataCompletenessCheck() \tag{60}$$

$$F_2 = ReferentialIntegrityCheck() \tag{61}$$

$$F_3 = ResourceCapacityCheck() \tag{62}$$

$$F_4 = TemporalWindowCheck() \tag{63}$$

$$F_5 = CompetencyEligibilityCheck() \tag{64}$$

$$F_6 = ConflictGraphCheck() \tag{65}$$

$$F_7 = GlobalConstraintCheck() \tag{66}$$

Expected feasibility output CSV:

Layer	Status	Violations
Data_Completeness	PASS/FAIL	count
Referential_Integrity	PASS/FAIL	count
Resource_Capacity	PASS/FAIL	count
Temporal_Window	PASS/FAIL	count
Competency_Eligibility	PASS/FAIL	count
Conflict_Graph	PASS/FAIL	count
Global_Constraints	PASS/FAIL	count

3.4 Stage-5: 16-Parameter Complexity Analysis

Expected complexity parameter calculation:

$$\Pi = (\pi_1, \pi_2, \ldots, \pi_{16}) \text{ where } \pi_i \in [0, 1]$$

Parameter definitions and expected ranges:

$$\pi_1 = \frac{\log_2(|C| \times |F| \times |R| \times |T| \times |B|)}{\log_2(10^6)} \in [0.1, 1.0] \tag{67}$$

$$\pi_2 = \frac{|constraints|}{|possible\_assignments|} \in [0.01, 0.3] \tag{68}$$

$$\pi_3 = 1 - \frac{1}{|F|} \sum_f \frac{|courses_f|}{|total\_courses|} \in [0.2, 0.8] \tag{69}$$

$$\vdots \tag{70}$$

$$\pi_{16} = \frac{\sigma^2(solution\_quality)}{\text{max\_variance}} \in [0.1, 0.9] \tag{71}$$

Expected complexity analysis CSV:

Parameter	Value	Category
Problem_Dimensionality	0.234	Low
Constraint_Density	0.156	Medium
Faculty_Specialization	0.678	High
⋮	⋮	⋮
Solution_Variance	0.445	Medium

3.5 Stage-6: Solver Family Output Validation

Expected solver outputs for each family:

3.5.1 PuLP Solver Output

Metric	Expected Range	Validation
Optimal_Value	$[0.8, 1.0]$	> 0.7
Solver_Status	OPTIMAL/FEASIBLE	status $\neq$ INFEASIBLE
Execution_Time	$[5, 300]$ seconds	< 600 seconds
Memory_Usage	$[50, 200]$ MB	< 512 MB

3.5.2 OR-Tools Solver Output

Metric	Expected Range	Validation
Objective_Value	$[0.85, 1.0]$	> 0.8
Search_Status	FEASIBLE/OPTIMAL	status = FEASIBLE
Branch_Count	$[100, 10000]$	< 50000
Wall_Time	$[10, 400]$ seconds	< 600 seconds

3.5.3 DEAP Evolutionary Output

Metric	Expected Range	Validation
Best_Fitness	$[0.75, 0.95]$	> 0.7
Generations	$[50, 500]$	< 1000
Population_Size	$[20, 100]$	fixed
Convergence_Rate	$[0.01, 0.1]$	> 0

3.5.4 PyGMO Global Optimization Output

Metric	Expected Range	Validation
Global_Best	$[0.8, 0.98]$	> 0.75
Islands_Count	$[4, 16]$	fixed
Migration_Rate	$[0.1, 0.3]$	parameter
Function_Evaluations	$[1000, 50000]$	< 100000

3.6 Stage-7: Output Validation

Expected validation metrics for schedule assignments:

$$V_{metrics} = (v_1, v_2, \ldots, v_{12})$$

$$v_1 = \text{Course Coverage Ratio} = \frac{|scheduled\_courses|}{|required\_courses|} \geq 0.95 \tag{72}$$

$$v_2 = \text{Faculty Workload Balance} = 1 - \frac{\sigma(faculty\_hours)}{\mu(faculty\_hours)} \geq 0.8 \tag{73}$$

$$v_3 = \text{Room Utilization} = \frac{|used\_rooms|}{|available\_rooms|} \in [0.7, 0.9] \tag{74}$$

$$v_4 = \text{Time Slot Efficiency} = \frac{|used\_slots|}{|total\_slots|} \in [0.6, 0.85] \tag{75}$$

$$\vdots \tag{76}$$

$$v_{12} = \text{Student Satisfaction} = \frac{|preferred\_assignments|}{|total\_assignments|} \geq 0.7 \tag{77}$$

4 FINAL OUTPUT GENERATION

4.1 Final Timetable Structure

Expected final_timetable.csv structure:

$$\mathcal{TT} = \{(assignment\_id, course, faculty, room, timeslot, batch, day, time, duration)\}$$

Constraints validation:

$$\forall a \in assignments : \tag{78}$$
$$course\_id \in valid\_courses \tag{79}$$
$$faculty\_id \in competent\_faculty(course\_id) \tag{80}$$
$$room\_id \in suitable\_rooms(course\_type) \tag{81}$$
$$timeslot\_id \in available\_slots \tag{82}$$
$$batch\_id \in enrolled\_batches(course\_id) \tag{83}$$
$$\neg\exists a' : conflicts(a, a') \text{ (no overlaps)} \tag{84}$$

Expected output metrics:

- Total assignments: $|assignments| = \sum_c \sum_b required\_sessions(c, b)$
- Hard constraint violations: $= 0$
- Soft constraint penalty: $\leq 0.3 \times max\_penalty$
- Faculty utilization: $\in [0.7, 0.9]$
- Room utilization: $\in [0.6, 0.85]$

5 TEST CASE GENERATION ALGORITHM

5.1 Master Generation Function

Listing 1: Test Case Generation Algorithm

```
def generate_comprehensive_test_case(seed: int, complexity_level: str) -> TestCa
    """
    Generate a complete, internally consistent test case
    Args:
        seed: Random seed for reproducibility
        complexity_level: 'simple', 'medium', 'complex'
    Returns:
        TestCase with all CSV files and expected outputs
    """
    random.seed(seed)
    np.random.seed(seed)

    # Scale parameters based on complexity
    scale_params = {
        'simple': {'students': 100, 'courses': 20, 'faculty': 15},
        'medium': {'students': 500, 'courses': 50, 'faculty': 30},
        'complex': {'students': 2000, 'courses': 100, 'faculty': 80}
    }

    params = scale_params[complexity_level]

    # 1. Generate base institutional data
    institution = generate_institution(seed)
```

```
departments = generate_departments(institution, count=random.randint(3, 8))
programs = generate_programs(departments, count_per_dept=3)
courses = generate_courses(programs, total_target=params['courses'])

# 2. Generate operational resources
shifts = generate_shifts(institution)
timeslots = generate_timeslots(shifts, slots_per_day=8)
faculty = generate_faculty(departments, total_target=params['faculty'])
rooms = generate_rooms(institution, capacity_for_students=params['students']
equipment = generate_equipment(rooms)

# 3. Generate student data and relationships
students = generate_students(programs, total_target=params['students'])
faculty_competency = generate_faculty_competency(faculty, courses)
constraints = generate_constraints(institution)

# 4. Calculate expected intermediate outputs
expected_batches = calculate_expected_batches(students, courses, rooms)
expected_enrollment = calculate_expected_enrollment(expected_batches, course
expected_prerequisites = normalize_prerequisites(courses)
expected_room_access = normalize_room_access(rooms, departments)

# 5. Calculate expected complexity analysis
expected_complexity = calculate_complexity_parameters(
    courses, faculty, rooms, timeslots, expected_batches
)

# 6. Generate expected solver outputs
expected_solver_outputs = {}
for solver in ['pulp', 'ortools', 'deap', 'pygmo']:
    expected_solver_outputs[solver] = simulate_solver_output(
        solver, courses, faculty, rooms, timeslots, expected_batches
    )

# 7. Generate expected final timetable
expected_final_timetable = generate_expected_timetable(
    expected_solver_outputs['ortools'],  # Use OR-Tools as baseline
    courses, faculty, rooms, timeslots, expected_batches
)

# 8. Calculate expected validation metrics
expected_validation = calculate_validation_metrics(expected_final_timetable)

return TestCase(
    input_csvs={
        'institutions': institution_to_csv(institution),
        'departments': departments_to_csv(departments),
        'programs': programs_to_csv(programs),
        'courses': courses_to_csv(courses),
```

```
            'shifts': shifts_to_csv(shifts),
            'timeslots': timeslots_to_csv(timeslots),
            'faculty': faculty_to_csv(faculty),
            'rooms': rooms_to_csv(rooms),
            'equipment': equipment_to_csv(equipment),
            'student_data': students_to_csv(students),
            'faculty_course_competency': competency_to_csv(faculty_competency),
            'constraints': constraints_to_csv(constraints)
        },
        expected_intermediate={
            'student_batches': expected_batches,
            'batch_course_enrollment': expected_enrollment,
            'course_prerequisites': expected_prerequisites,
            'room_department_access': expected_room_access,
            'feasibility_check': generate_feasibility_expectations(),
            'complexity_analysis': expected_complexity,
            'solver_outputs': expected_solver_outputs
        },
        expected_final={
            'schedule_assignments': expected_final_timetable,
            'validation_metrics': expected_validation
        },
        metadata={
            'seed': seed,
            'complexity': complexity_level,
            'generated_at': datetime.now(),
            'total_students': params['students'],
            'total_courses': params['courses'],
            'total_faculty': params['faculty']
        }
    )
```

5.2 Validation Framework

Listing 2: Test Case Validation Framework

```
class TestCaseValidator:
    """Rigorous validation of test case generation and execution results"""

    def validate_complete_pipeline(self, test_case: TestCase, actual_results: Di
        """
        Comprehensive validation of entire pipeline execution
        """
        report = ValidationReport()

        # 1. Input Data Validation
        report.input_validation = self.validate_input_consistency(test_case.inpu

        # 2. Stage-by-Stage Validation
        for stage_num in range(1, 8):
```

```
            stage_key = f'stage_{stage_num}'
            expected = test_case.expected_intermediate.get(stage_key)
            actual = actual_results.get(stage_key)

            if expected and actual:
                report.stage_validations[stage_num] = self.validate_stage_output
                    stage_num, expected, actual, tolerance=0.05
                )

        # 3. Final Output Validation
        report.final_validation = self.validate_final_timetable(
            test_case.expected_final['schedule_assignments'],
            actual_results.get('final_timetable'),
            test_case.input_csvs
        )

        # 4. Performance Validation
        report.performance_validation = self.validate_performance_metrics(
            actual_results.get('performance_metrics'),
            expected_limits=test_case.metadata
        )

        return report

    def validate_mathematical_consistency(self, test_case: TestCase) -> bool:
        """
        Validate that generated test case satisfies all mathematical constraints
        """
        # Resource capacity constraints
        total_student_hours = sum(
            batch['student_count'] * len(batch['courses']) * 3  # 3 hours per co
            for batch in test_case.expected_intermediate['student_batches']
        )

        total_faculty_capacity = sum(
            faculty['max_hours_per_week']
            for faculty in test_case.input_csvs['faculty']
        )

        # Validate: total demand <= total supply with 20% buffer
        if total_student_hours > total_faculty_capacity * 0.8:
            return False

        # Room capacity constraints
        max_concurrent_students = max(
            sum(batch['student_count']
                for batch in test_case.expected_intermediate['student_batches']
                if batch.get('preferred_shift') == shift_id)
            for shift_id in [s['shift_id'] for s in test_case.input_csvs['shifts
```

```python
    )

    total_room_capacity = sum(
        room['capacity']
        for room in test_case.input_csvs['rooms']
        if room['room_type'] in ['CLASSROOM', 'LABORATORY']
    )

    # Validate: peak demand <= total room capacity
    if max_concurrent_students > total_room_capacity:
        return False

    return True
```

6 IMPLEMENTATION SPECIFICATIONS

6.1 Test Case Storage Structure

Each generated test case creates the following directory structure:

```
test_cases/
 tc_{seed}_{complexity}/
    inputs/
       institutions.csv
       departments.csv
       programs.csv
       courses.csv
       shifts.csv
       time_slots.csv
       faculty.csv
       rooms.csv
       equipment.csv
       student_data.csv
       faculty_course_competency.csv
       constraints.csv
    expected_outputs/
       stage_2_batches.csv
       stage_2_enrollment.csv
       stage_3_prerequisites.csv
       stage_3_room_access.csv
       stage_4_feasibility.csv
       stage_5_complexity.csv
       stage_6_pulp_output.csv
       stage_6_ortools_output.csv
       stage_6_deap_output.csv
       stage_6_pygmo_output.csv
       stage_7_validation.csv
       final_timetable.csv
    metadata.json
    validation_rules.json
```

6.2 Reproducibility Guarantees

The framework ensures complete reproducibility through:

$$TestCase(seed_i) = TestCase(seed_i) \quad \forall \text{ executions} \tag{85}$$
$$||ExpectedOutput - ActualOutput|| < \epsilon \quad \text{for correctly implemented stages} \tag{86}$$
$$ValidationScore(TestCase) \geq 0.95 \quad \text{for all generated cases} \tag{87}$$

7 QUALITY ASSURANCE METRICS

7.1 Test Case Coverage Matrix

Component	**Simple**	**Medium**	**Complex**	**Coverage Target**
Data Consistency	100%	100%	100%	All relationships validated
Constraint Satisfaction	95%	90%	85%	Most constraints feasible
Algorithm Convergence	100%	95%	90%	Solutions found
Performance Bounds	100%	100%	95%	Within resource limits
Output Completeness	100%	100%	100%	All required fields

7.2 Validation Tolerance Specifications

$$\text{Numerical Tolerance} : |\text{expected} - \text{actual}| < 10^{-3} \tag{88}$$
$$\text{Count Tolerance} : |\text{expected\_count} - \text{actual\_count}| \leq 1 \tag{89}$$
$$\text{Percentage Tolerance} : |\text{expected\_ratio} - \text{actual\_ratio}| < 0.05 \tag{90}$$
$$\text{Time Tolerance} : |\text{expected\_time} - \text{actual\_time}| < 10\% \tag{91}$$
$$\text{Boolean Accuracy} : \text{exact match required} \tag{92}$$

8 CONCLUSION

This rigorous mathematical framework provides a complete foundation for generating deterministic, validated test cases for the scheduling engine. The system ensures:

- **Deterministic Generation**: Same seed produces identical test cases
- **Mathematical Consistency**: All generated data satisfies relational constraints
- **Comprehensive Coverage**: All 12 input files, 7 processing stages, and final output validated
- **Scalable Complexity**: Support for simple, medium, and complex problem instances

- **Quantitative Validation**: Precise tolerance specifications for all output comparisons
- **Performance Verification**: Resource usage and execution time bounds

This framework enables exhaustive testing of the scheduling engine, ensuring robust validation before deployment and providing clear benchmarks for algorithmic improvements.