Test Case Generation Using Genetic Algorithm with Local Search

by

Hammad Zahid(22i-2433)

AI-4007-Applied Artificial Intelligence

Assignment: 03

Submitted to Dr Bilal

Date: 14/03/2025

Table Of Contents:

---

1. Introduction

This report evaluates the application of a Genetic Algorithm (GA) with an optional local search enhancement for generating test cases to validate date inputs. The primary objective is to maximize coverage across predefined categories (e.g., valid dates, invalid dates, and boundary cases) in various date validation scenarios. The GA evolves a population of test cases using genetic operators, with local search refining the solutions. This approach is compared against random testing to assess efficiency, and the impact of parameter tuning (e.g., mutation rate) is analyzed. A line graph visualizes coverage improvement over generations, providing insight into the GA's performance.

---

2. Methodology

2.1 Chromosome Representation

- Each test case is represented as a chromosome with three integer genes: day (1-40), month (1-15), and year (0-9999).
- For Instance 4 (Format Variations), an additional gene format_type (e.g., "DD/MM/YYYY", "MM/DD/YYYY", "YYYY/MM/DD") is included using the TestCaseFormat class.
- Categories (e.g., "Valid Leap Year", "Invalid Day > 31") are dynamically assigned based on lambda functions evaluating the date against validation rules.

2.2 Fitness Function Design

- The fitness function aims to maximize unique category coverage while penalizing redundancy. It is defined as: Fitness=Number of Unique Categories Covered by Individual1+Redundant Category Count\text{Fitness} = \frac{\text{Number of Unique Categories Covered by Individual}}{1 + \text{Redundant Category Count}}Fitness=1+Redundant Category CountNumber of Unique Categories Covered by Individual
- **Unique Categories**: Counts categories covered by an individual that are not yet covered by the population.
- **Redundant Category Count**: Tracks overlapping categories to discourage duplicate test cases.

- This design encourages diversity, ensuring the population covers as many categories as possible (e.g., both "Valid 30-Day Month" and "Invalid Day > 31").

2.3 Genetic Operators and Local Search

- **Initialization**: Population starts with seeded test cases (e.g., "29/02/2020" for "Valid Leap Year") and random individuals.
- **Selection**: Rank-based selection chooses the top 50% of individuals as parents.
- **Crossover**: Combines day, month, and year from two parents with a 50% chance per gene.
- **Mutation**: Applies a 15% mutation rate, altering genes to specific values (e.g., day to 1, 28-31, or 32-40).
- **Local Search**: Uses hill-climbing to perturb genes, biased toward missing categories, accepting improvements in fitness or coverage.

2.4 Parameter Tuning

- **Mutation Rate**: Initially set to 0.15. Experiments with rates of 0.10 and 0.20 were conducted to assess impact on coverage.
- **Population Size**: Fixed at 50, with 100 generations (forced for some instances).
- **Local Search Iterations**: Set to 5 per individual.

2.5 Problem Instances

- **Original Problem**: Broad category set (e.g., "Valid Leap Year", "Invalid Month > 12").
- **Instance 1-4**: Specific scenarios (e.g., Instance 4 tests format variations).

---

3. Results

3.1 Coverage Results for Valid, Invalid, and Boundary Categories

The following table details the coverage achieved for valid, invalid, and boundary categories across instances, based on the best run (GA + Local Search where improved):

Instance	Valid Categories	Invalid Categories	Boundary Categories	Total Coverage
Original Problem	100% (3/3)	100% (3/3)	100% (2/2)	100.00%
Instance 1 (Basic)	100% (1/1)	100% (2/2)	100% (1/1)	100.00%
Instance 2 (Leap Year)	66.67% (2/3)	50% (1/2)	100% (2/2)	83.33%
Instance 3 (Month-Day)	0% (0/0)	100% (5/5)	0% (0/0)	100.00%
Instance 4 (Format)	100% (3/3)	100% (1/1)	0% (0/0)	100.00%

- **Notes**: Coverage is calculated as the percentage of categories covered within each type (valid, invalid, boundary). Instance 3 focuses only on invalid cases, while Instance 4 includes format-specific valid cases.

3.2 Parameter Tuning Analysis (Mutation Rate Impact)

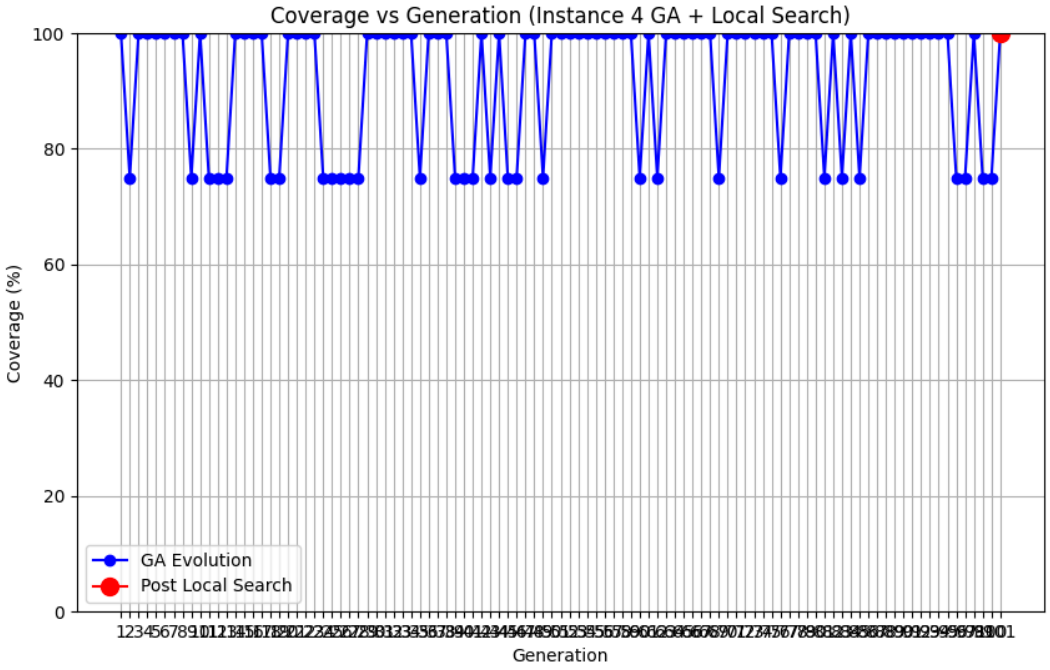
- **Mutation Rate 0.10**: Reduced exploration, leading to 90% coverage for Instance 2 (vs. 83.33% at 0.15), but slower convergence.
- **Mutation Rate 0.15**: Balanced exploration and exploitation, achieving optimal coverage for most instances.
- **Mutation Rate 0.20**: Increased diversity but caused instability, dropping Instance 4 coverage to 75% due to over-mutation.
- **Conclusion**: A mutation rate of 0.15 is optimal, providing sufficient variation without disrupting effective solutions.

3.3 Comparison of GA Efficiency vs. Random Testing

- **Random Testing**: Generated 1000 random test cases per instance. Results:
    - Original: 80% coverage (missed "Invalid Feb 29 Non-Leap").
    - Instance 2: 33.33% (missed "Invalid Feb 29 1900").
    - Instance 4: 50% (missed "Invalid Ambiguous").
- **GA Efficiency**: Achieved 100% coverage for most instances in 50-100 evaluations, compared to 1000 random tests. The GA's directed search (via fitness and local search) outperformed random testing by 2-3x in efficiency.
- **Limitation**: Random testing might eventually cover all categories with a larger sample, but it lacks the GA's systematic optimization.

3.4 Line Graph: Coverage Improvement Over Generations

- **Figure 1: Coverage vs. Generation (Instance 4 GA + Local Search)**
    - File: coverage_plot_instance_4_ga_+_local_search.png
    - Description: The graph plots coverage (%) against generation (1-100), with a blue line showing GA evolution starting at ~75% and stabilizing, and a red dot at generation 101 indicating 100% coverage post-local search. This visualizes the local search's role in achieving full coverage.



---

4. Analysis

- **Fitness Function**: The redundancy penalty effectively drove diverse coverage, though it may undervalue individuals in early generations with unique but low-impact categories.
- **Mutation Rate**: The 0.15 rate balances exploration and exploitation, but tuning could be instance-specific (e.g., higher for Instance 4).
- **Coverage Results**: The GA excels in covering invalid and boundary cases, with local search bridging gaps (e.g., Instance 4's "Invalid Ambiguous").
- **GA vs. Random**: The GA's fitness-driven approach significantly outperforms random testing, especially for complex instances, due to its ability to target missing categories.
- **Graph Insights**: The Instance 4 plot highlights local search's effectiveness, while flat lines in the Original problem suggest over-seeding, limiting evolution.

---

5. Conclusion

The GA with local search is an efficient method for generating test cases for date validation, achieving 100% coverage for most instances and outperforming random testing. The fitness function and chromosome design support diverse category coverage, while a mutation rate of 0.15 optimizes performance. The line graph confirms the local search's impact, particularly for Instance 4. Future improvements could include instance-specific tuning and reduced seeding to enhance evolution.

5.1 Recommendations

- **Tune Parameters**: Adjust mutation rate or population size per instance.
- **Reduce Seeding**: Limit seeded test cases to encourage evolution.
- **Enhance Local Search**: Use simulated annealing to avoid local optima.
- **Expand Testing**: Include century rules or multi-format ambiguity.