Evaluating the Trade-offs of Diversity-Based Test Prioritization
An Experiment

Lemon Ginger Thesis

Chi Hong Chao
Ranim Khojah

Supervisor: Francisco Gomes de Oliveira Neto

Introduction

- In this study, we aim to:
  - Perform an **efficient and exhaustive** investigation of a range of Diversity Test Prioritization Techniques through an experimental study
  - **Critically evaluate** the cost-effectiveness of these techniques on three levels of testing.
- Contributions:
  - We **uncover new knowledge** by identifying the hindrances of Semantic Similarity in comparison with string distances.
  - We **synthesized** all our **trade-off discussion** into recommendations
  - Implement an adaptable workflow to run prioritization techniques in a project, **reusable** for practitioners or future studies.
  - Implement Semantic Similarity, a prioritization technique that uses the meaning of words to compare test cases, also **reusable** for future work or practitioners.

Background (An Example)

DBT(Lexically):

```
1  test_get_ATM();
2  test_insert_invalid_card();
3  test_use_expired_card();
4  test_deposit();
5  test_insert_valid_card();
6  test_withdraw();
```

```
1  test_deposit();
2  test_withdraw();
3  test_insert_invalid_card();
4  test_use_expired_card();
5  test_insert_valid_card();
6  test_get_ATM();
```

DBT
(Semantically):

```
1  test_get_ATM();
2  test_insert_valid_card();
3  test_insert_invalid_card();
4  test_deposit();
5  test_use_expired_card();
6  test_withdraw();
```

Research questions

RQ1: How do DBT perform in terms of ***coverage*** ?
RQ2: To what extent does each DBT uncover ***failures***?

Effectiveness

RQ3: How ***long does it take*** to execute DBT on different levels of testing?

Cost

RQ4: How do **different levels of testing** affect the *diversity* of a test suite?

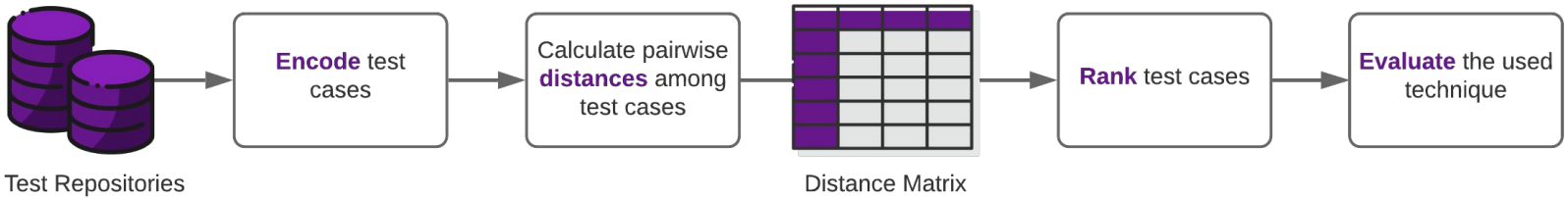
Levels of Testing

Methodology

- **Fractional Factorial** Experiment

Technique/ Level	Unit Level	Integration Level	System Level
SS			X
Jaccard Index	X	X	X
Levenshtein	X	X	X
NCD	X	X	X

- The **workflow** followed for *each technique* in the experiment



Data Collection

- APFD - Average Percentage of Failures Detected - How early the prioritized test suite detects failures
- Coverage - Traceability information → corresponding requirement of test
- Execution Time - How long it takes for a technique to rank a test suite from beginning to end

Unit - Defects4J, 7 Open Source Projects

1. APFD
   1.1. Get each version for all faults in project
   1.2. Extract test methods
   1.3. Aggregate results using mean
   1.4. Calculate failures at different cutoffs
2. Time
   2.1. Once per version, for each project
   2.2. Differing versions for each project, ranging from 18 to 106

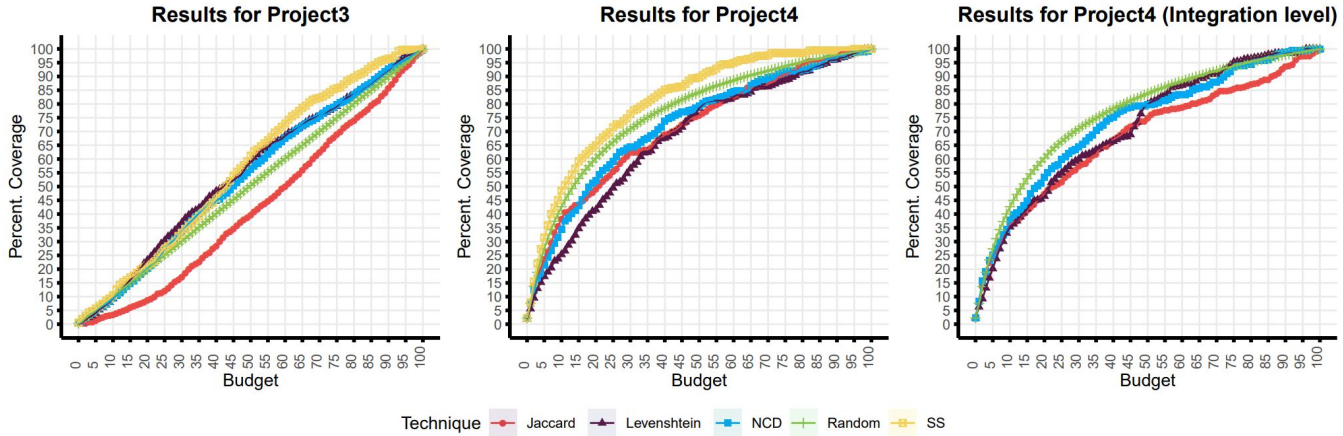
Integration/ System - 2 Industry Partners, 4 Projects

1. Coverage
   1.1. If a test is mapped to >= 1 system requirement
   1.2. Get list of tests linked with their requirements
   1.3. Calculate total covered requirements using the ranked tests and features linked to tests with an algorithm
2. APFD
   2.1. Selected 115 out of 669 builds
   2.2. Results (execution history) for builds collected to know which tests uncovered which failures
3. Time
   3.1. 10 times per technique, for each project

Results and Analysis

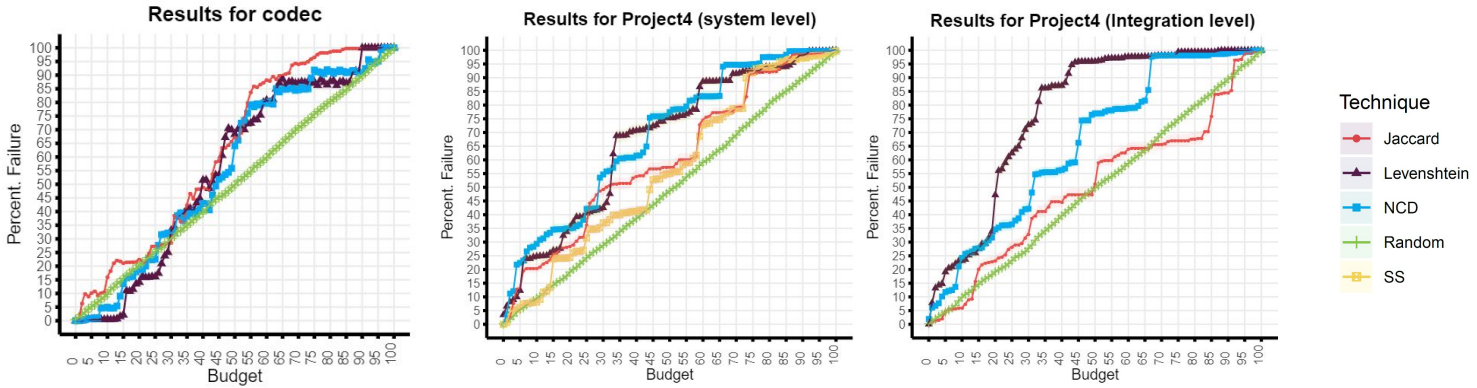
*RQ1: How do DBT differ on system and integration levels in terms of **coverage**?*

- **System level:** Semantic Similarity covers the most features.
- **System+Integration levels:** NCD and Levenshtein's coverage are similar.
  Jaccard's coverage rate is low.



*RQ2: How do DBT differ in terms of **failure detection** on different Levels of testing?*

- No technique consistently finds most failures on the 3 LoTs.
- Greater distinction between *most* DBT and Random as budget increases up until 80%.
- Statistically speaking, SS has SSD with all DBT, *except* with Jaccard.

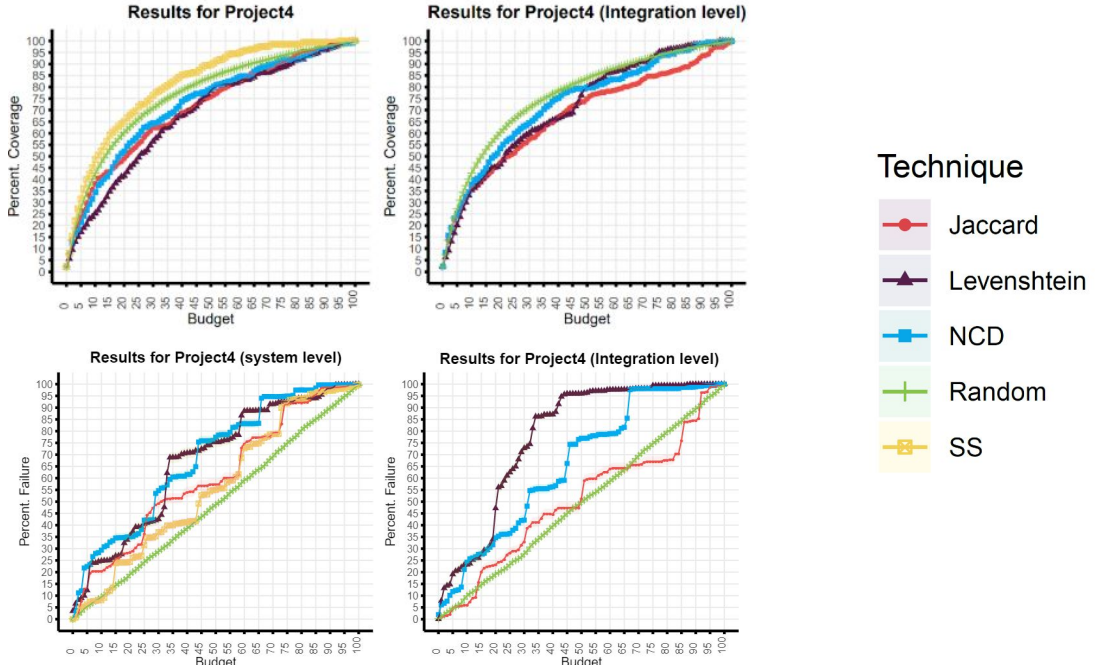


*RQ3: **How long** does it take to execute each technique on different level of testing?*

Fastest ├ Random - Jaccard - SS - NCD - Levenshtein ┤ **Slowest**

*RQ4: How do different **levels of testing** affect the **prioritization** of a test suite?*

- Coverage:
- Failure (APFD):
- Execution Time: All techniques *but Levenshtein* are faster **on system level**.



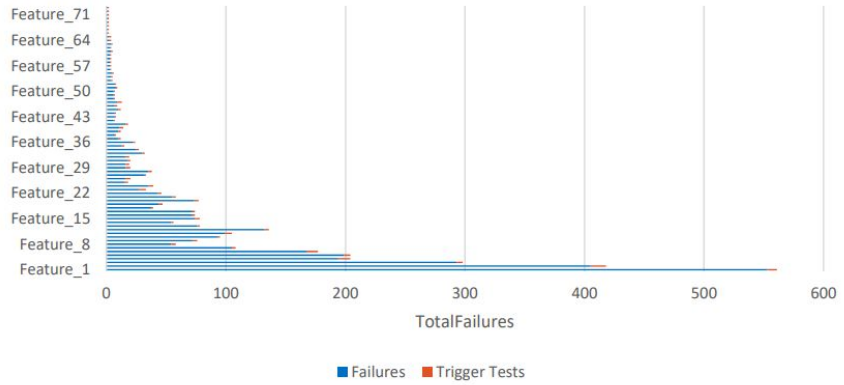
Discussion - Recommendations

Technique	Recommended LoT	General Trade-offs
Jaccard	Unit	Speed = **Fast++** Coverage = System Level Bad Failure = **Unit Good**, System Moderate, Integration Bad
Levenshtein	Integration	Speed = **Slow+++ - Is like Snail** - compared to itself, faster on unit level Coverage = System Level Bad Failure = **Integration Good**, Unit Bad
NCD	Unit, System	Speed = **Slow on Integration**, but faster than Levenshtein Coverage = System Level Bad Failure = Good across all LoT - ***Consistent***
SS	System	Speed = Fast+ Coverage = **Good+** Failure = System Moderate, ***only applicable to System Level***
Random	When <= 20% or >= 90% budget, all LoT	Speed = **Fast+++++++++ - Is speed itself** Coverage = Good Failure = **Pretty bad, depends on budget + naturally diverse test suite**

Intermission

In-depth Discussions

- Random is recommended to be used if budget is < 20% or > 90%, due to low SSD and speed.
- If **failure detection is priority**, the budget should be chosen by studying the failure history to understand the nature of the failure distribution.
- Both the **number of tests** and the **size** of the tests should be considered when deciding which DBT to run. Levenshtein is generally too expensive and unreliable.
- For system level tests, unsound data like long, unreadable strings should be cleaned regardless of DBT for a better performance.



Conclusions and **Future Work**

Strengths	Weaknesses
11 projects studied across all levels	1 test subject for failures on System/Integration
Realistic failures used on all 3 levels	Low control of randomness for techniques
The comparison between SS and other DBT are novel	Uneven comparison - sampling strategy limitation - Not all techniques executed on all projects

Possible future work:

- SS on unit level and integration level
- Effects of test structures on different techniques

Thank you very much bye bye