International Joint Conference on Neural Networks IJCNN @ WCCI 2018, 8 - 13 July

A new word embedding approach to evaluate potential fixes for automated program repair

Leonardo Afonso Amorim, Mateus F. Freitas, Altino Dantas, Eduardo F. de Souza, Celso G. Camilo-Junior and Wellington S. Martins

UFG

INSTITUTO DE INFORMÁTICA UFG

FAPEG
Fundação de Amparo à Pesquisa do Estado de Goiás

I4Soft
Intelligence for Software

Agenda

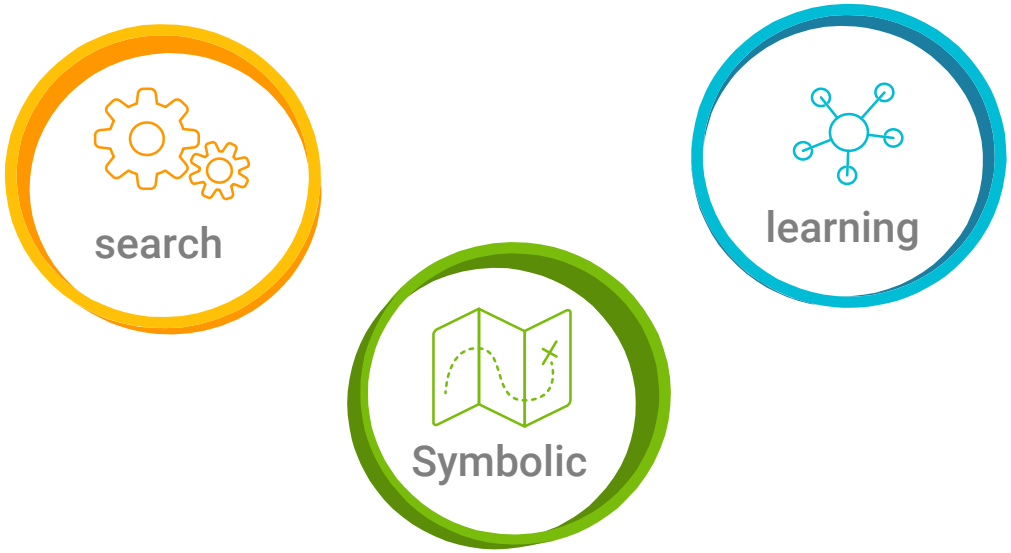
- Introduction
- Related work
- Approach
- Experiments
- Remarks

Introduction

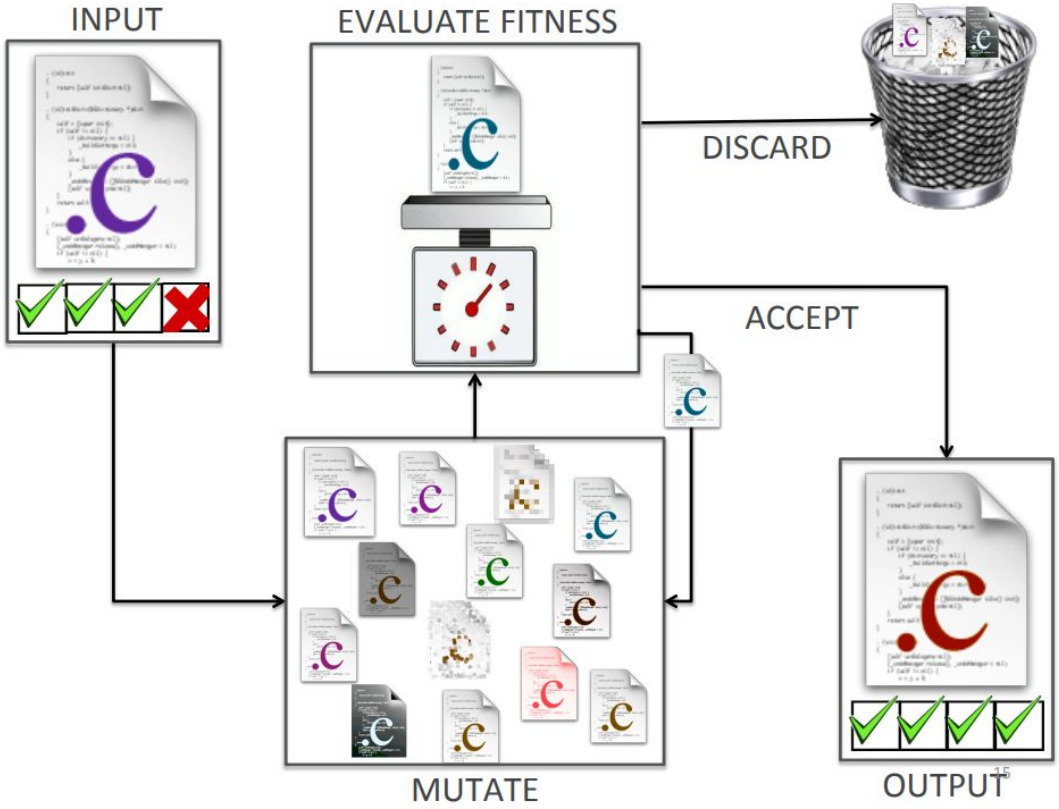
Context

- Software maintenance is a **costly** task;
- Software debugging often is a **manual and complex** issue;
- **Automated** program repair (APR);
- Typically, APR methods use **test cases** to automatically evaluate potential fixes.

Introduction



Introduction



Introduction

Problem

- Test suites could be not complete enough in real-word scenarios;
- Metrics based on test cases produce plateaus;
- High computational cost by executing original and variant programs.

Introduction

Our solution

This work proposes applying Word2Vec, a word embedding model, to improve the repair evaluation process based on the naturalness obtained from a corpus of known fixes.

Related Work

- **[2016] - T. Ji, L. Chen, X. Mao and X. Yi**
  Use syntax **similarity and dissimilarity** to define a threshold for acceptable patches;

- **[2016] - Z. Zojaji, B. T. Ladani and A. Khalilian**
  Check a model of well-defined **software properties** to measure the variant's fitness;

- **[2016, 2017] - S. H. Tan et al. | M. Jiang et al.**
  Compute some relation over **negative or positive** tests from a test suite (or metamorphic tests).

Background

- **Word embedding**
  - Language modeling and feature learning techniques in Natural Language Processing (NLP);
  - Words or phrases from the vocabulary are mapped to a continuous space.
- **Word2Vec**
  - Learns continuous word embeddings from plain text in an entirely unsupervised way;
  - Neural network trained with streams of n-grams of words so as to predict the n-th word, given words [1,..., n−1] or the other way around.

Approach

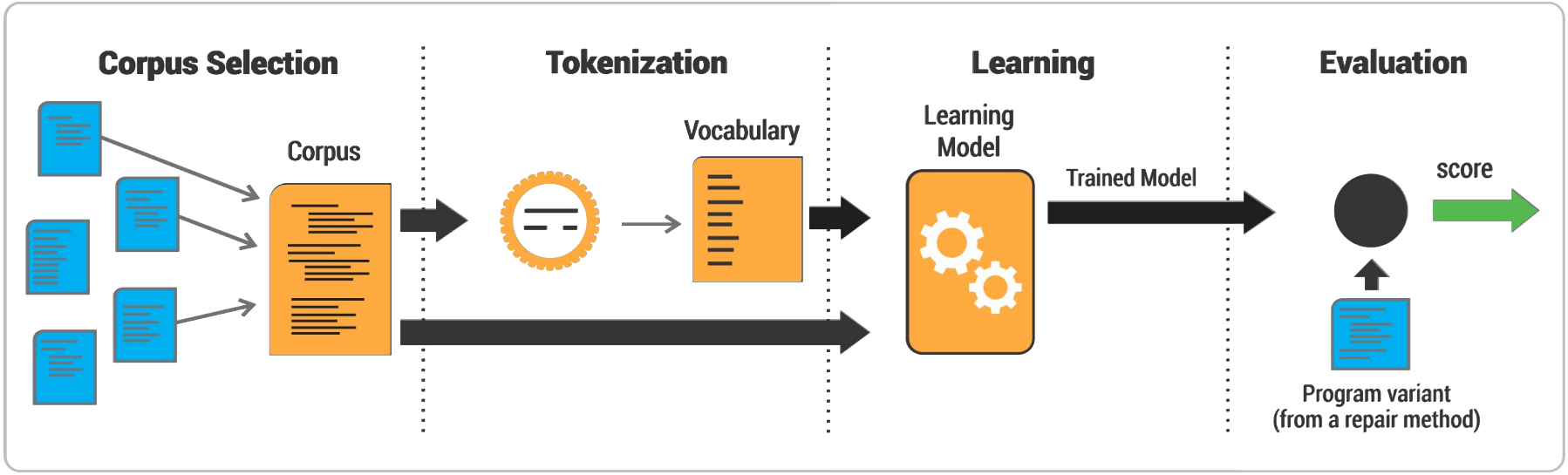


Figure 1: Proposal's flowchart. Produced by authors, 2018.

Experiments - Design

- **We defined two metrics:**
  - ***Prob*** (based on the softmax layer);
  - ***Dist*** (continuous tokens representation);
- **Introclass benchmark;**
- **Generation of variants by:**
  - Deleting, inserting and swapping;
  - Various levels of perturbation.

Word2vec parameters

- Tokenization by **lines**;
- Size of continuous vector: 50;
- Window context: 1.

RQ - Are the metrics Prob and Dist able to capture the loss of naturalness of variants?

Experiments - Quantitative evaluation

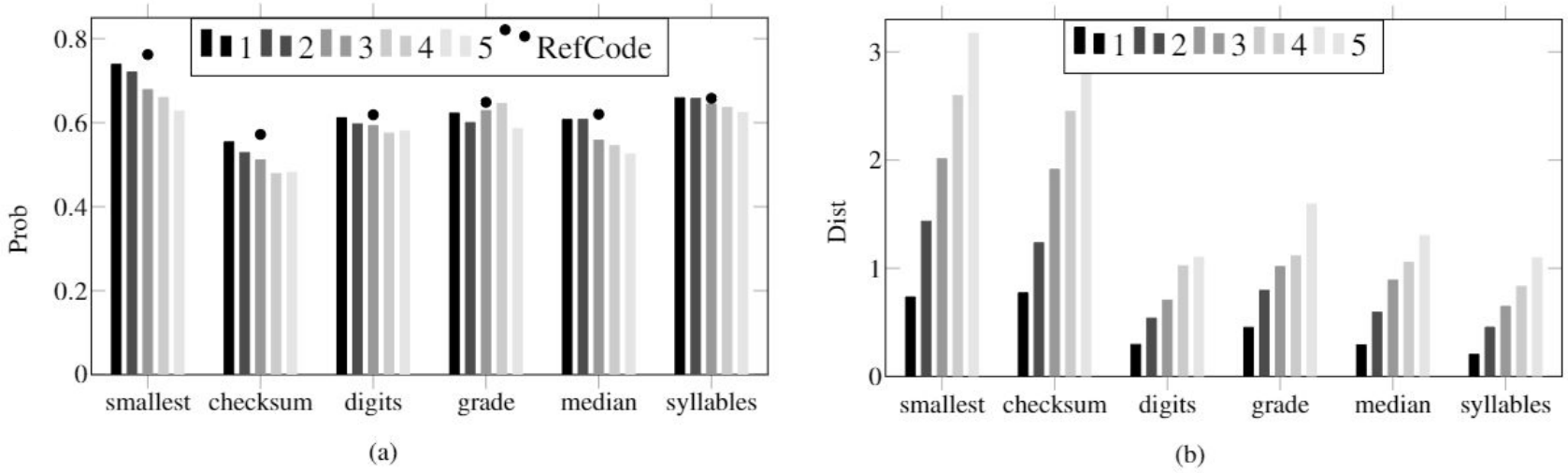


Figure 2: *Prob* and *Dist* values after applying **Delete operator** in each original program. Produced by authors, 2018.

Experiments - Quantitative evaluation

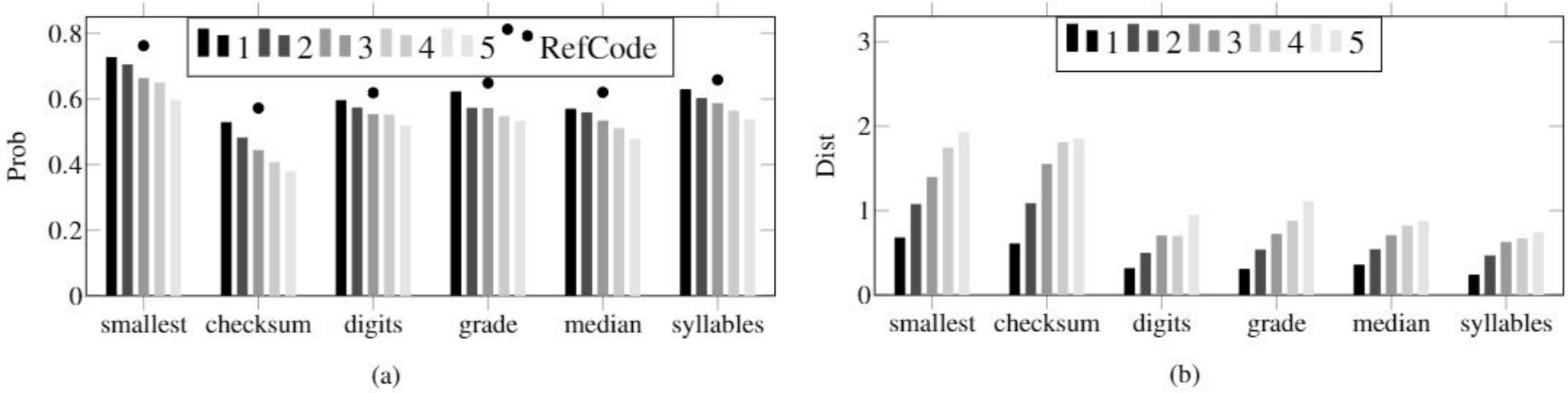


Figure 3: *Prob* and *Dist* values after applying **Insert operator** in each original program. Produced by authors, 2018.

Experiments - Quantitative evaluation

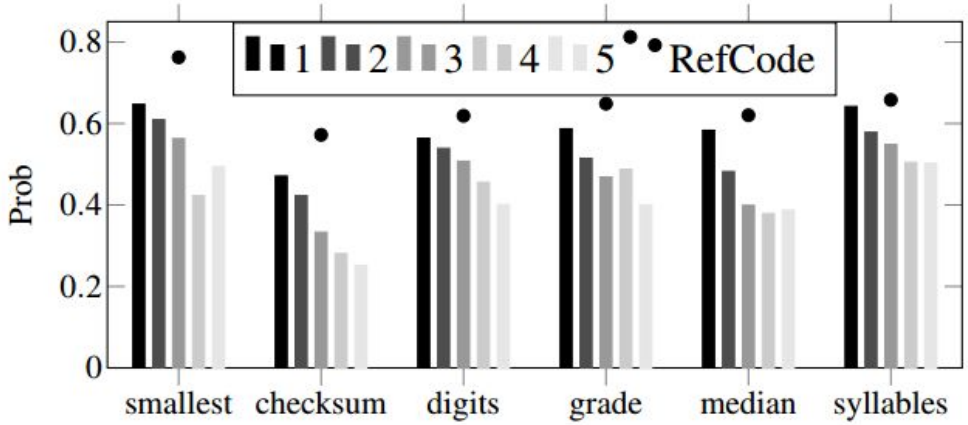


Figure 4: *Dist* values after applying **Swap operator** in each original program. Produced by authors, 2018.

Examples of evaluating variants

GenProg score = 0
Prob = 0.59
Dist = 1.09

```c
#include <stdio.h>
#include <math.h>
int main() {
   int a, b, c, d;
   printf("Please enter 4 numbers separated by spaces>");
   scanf("%d %d %d %d", &a, &b, &c, &d);
   if ((a <= b) && (a <= c) && (a <= d)) {
      printf("%d is the smallest\n", a);
      return (0);
   } else if((b <= a) && (b <= c) && (b <= d)) {
      printf("%d is the smallest\n", b);
      return (0);
   } else if((c <= a) && (c <= b) && (c <= d)) {
      printf ("%d is the smallest\n", c) ;
      return (0);
   } else if((d <= a) && (d <= b) && (d <= c)) {
      printf ("%d is the smallest\n", d);
      return (0);
   } else {
      printf ("%d is the smallest\n", a);
   }
}
```

variant A for smallest

GenProg score = 0
Prob = 0.74
Dist = 0.65

```c
#include <stdio.h>
#include <math.h>
int main() {
   int a, b, c, d;
   printf("Please enter 4 numbers separated by spaces>");
   scanf("%d %d %d %d", &a, &b, &c, &d);
   if ((a <= b) && (a <= c) && (a <= d)) {
   } else if ((b <= a) && (b <= c) && (b <= d)) {
      printf ("%d is the smallest\n", b );
   } else if ((c <= a) && (c <= b) && (c <= d)) {
      printf("%d is the smallest\n", c);
   } else if (( d <= a ) && ( d <= b ) && (d <= c)) {
      printf("%d is the smallest\n", d);
   }
}
```

variant B for smallest

```c
#include <stdio.h>
#include <math.h>
int main() {
   int a, b, c, d;
   printf("Please enter 4 numbers separated by spaces>");
   scanf("%d %d %d %d", &a, &b, &c, &d);
   if((a <= b) && (a <= c) && (a <= d)) {
      printf("%d is the smallest\n", a);
   } else if((b <= a) && (b <= c) && (b <= d)) {
      printf("%d is the smallest\n", b);
   } else if((c <= a) && (c <= b) && (c <= d)) {
      printf("%d is the smallest\n", c );
   } else if((d <= a) && (d <= b) && (d <= c)) {
      printf ("%d is the smallest\n", d);
   }
   return (0);
}
```

Correct program for smallest

Figure 5: Variants from the smallest program and their Dist and Prob values. Produced by authors, 2018.

Remarks

- Both proposed metrics **Prob** and **Dist** showed the proposal is able to differentiate variants with **different levels of perturbation**;
- The results corroborate our assumptions the word embedding is able to **measure naturalness** of program variants (static analysis).

Acknowledgement

We would like to thanks FAPEG for supporting this presentation.

Thanks!

Doubts/Suggestions