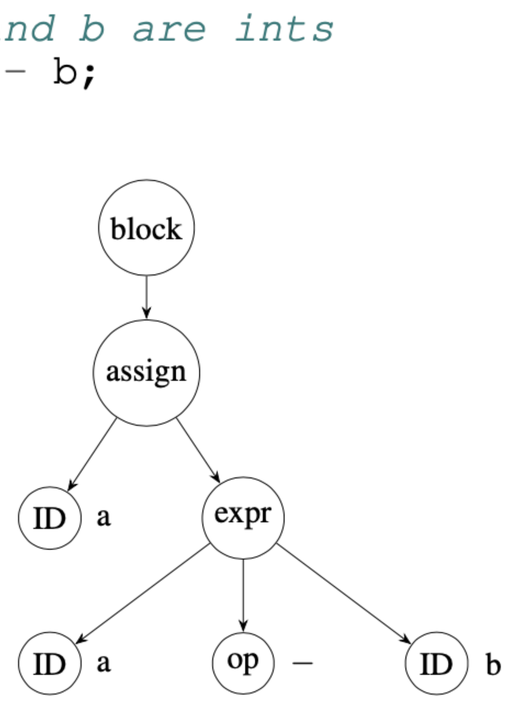
Graph Neural Networks For Mapping Variables Between Programs

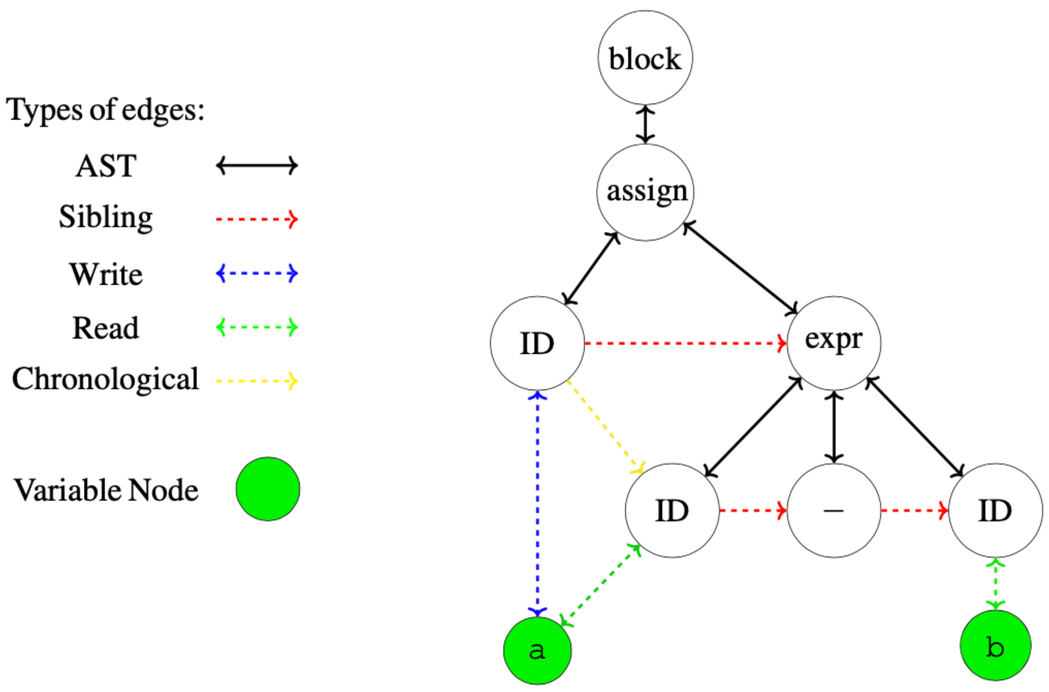
Pedro Orvalho [1], Jelle Piepenbrock [2,3], Mikoláš Janota [3], and Vasco Manquinho [1]

[1] INESC-ID, IST, Universidade de Lisboa, Portugal [2] Radboud University Nijmegen, The Netherlands

[3] Czech Technical University in Prague, Czechia

```
1  { // a and b are ints
2    a = a - b;
3  }
```





1. Motivation

- Comparing two programs is highly challenging;
- A relation between two programs' sets of variables is required;
- Mapping variables between both programs is useful for *program equivalence*, *program repair, etc.*

Listing 1: A semantically correct student's implementation.

```
1  int main(){
2      int n, i;
3      scanf("%d", &n);
4      for(i = 1; i <= n; i++){
5          printf("%d\n", i);
6      }
7      return 0;
8  }
```

Listing 2: A semantically incorrect student's implementation since the variable `j` in the `main` function is not initialized.

```
1  void loop(int j, int l){
2    while (l >= j){
3      printf("%d\n", j);
4      ++j;
5    }
6  }
7  int main(){
8    int j, l;
9    scanf("%d", &l);
10   loop(j, l);
11   return 0;
12 }
```

Figure 1: Two implementations for the IPA of printing all the natural numbers from 1 to a given number n. The program in Listing 2 is semantically incorrect since the variable `j`, which is the variable being used to iterate over all the natural numbers until the number `l`, is not being initialized, i.e., the program has a bug of *missing expression*. The mapping between these programs' sets of variables is {`n : l`; `i : j`}.

2. Contributions

- A program representation that is agnostic to the names of the variables;
- We use GNNs for mapping variables between programs;

3. Program Representation

- We represent programs as *directed graphs* so the information can propagate in both directions in the GNN [1].
- These graphs are based on the programs' *abstract syntax trees* (ASTs).
- *Novelty:* **a unique variable node for each variable in the program** and connect each variable's occurrence to its unique node.
- Prior work on representing programs as graphs [2] **use different nodes for each variable occurrence** and **take into consideration the variable identifier** in the program representation.
- Our GNN is a *relational graph convolutional neural network* (RGCN):

$$\mathbf{x}'_i = \mathbf{\Theta}_{\text{root}} \cdot \mathbf{x}_i + \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_r(i)} \frac{1}{|\mathcal{N}_r(i)|} \mathbf{\Theta}_r \cdot \mathbf{x}_j,$$

4. Experiments

4.1 Dataset

- We used C-Pack-IPAs [3], a set of students' programs from two academic year, to evaluate our work.
- As training data, **we generated a dataset of pairs of correct/incorrect programs** using MultIPAs [4].
- Incorrect program have one of these types of bugs: *wrong comparison operator* (WCO), *variable misuse* (VM), and *missing expression* (ME).
- *First year* submissions are **divided into a training and validation set**.
- The *second year* was used for the **evaluation set**, consists only of new submissions, simulating a new academic year.

4.2 Training and Evaluation

- *Evaluation metrics:* totally correct mappings and the overlap coefficient

Evaluation Metric	Buggy Programs			
	WCO Bug	VM Bug	ME Bug	All Bugs
# Correct Mappings	87.38%	81.87%	79.95%	82.77%
Avg Overlap Coefficient	96.99%	94.28%	94.51%	95.05%
# Programs	1078	1936	1152	4166

4.3 Program Repair

- We tried to fix each pair of incorrect/correct programs in the evaluation dataset by passing each pair to every repair method: Verifix [5], Clara [6], and our repair approach based on the GNN's variable mappings.
- The baseline is using variables generated based on a uniform distribution.
- The **GNN correctly maps 83% of the evaluation dataset;**
- We leverage the variable mappings to perform automatic program repair;
- While the current state-of-the-art on program repair can only repair about 72% of the evaluation dataset due to structural mismatch errors, **our approach, based on variable mappings, is able to fix 88.5%.**

Repair Method	Buggy Programs				Not Succeeded	
	WCO Bug	VM Bug	ME Bug	All Bugs	% Failed	% Timeouts (60s)
Baseline	618 (57.33%)	1187 (61.31%)	287 (24.91%)	2092 (50.22%)	0 (0.0%)	**2074 (49.78%)**
VERIFIX	555 (51.48%)	1292 (66.74%)	741 (64.32%)	2588 (62.12%)	**1471 (35.31%)**	107 (2.57%)
CLARA	722 (66.98%)	1517 (78.36%)	764 (66.32%)	3003 (72.08%)	1153 (27.68%)	10 (0.24%)
GNN	**992 (92.02%)**	**1714 (88.53%)**	**981 (85.16%)**	**3687 (88.5%)**	0 (0.0%)	479 (11.5%)

References

[1] P. Orvalho, J. Piepenbrock, M. Janota, and V. Manquinho. Project Proposal: Learning Variable Mappings to Repair Programs. In 7th Conference on Artificial Intelligence and Theorem Proving, AITP 2022.

[2] M. Allamanis, M. Brockschmidt, and M. Khademi. Learning to represent programs with graphs. ICLR 2018.

[3] P. Orvalho, M. Janota, and V. Manquinho. C-Pack of IPAs: A C90 Program Benchmark of Introductory Programming Assignments. arXiv. 2022.

[4] P. Orvalho, M. Janota, and V. Manquinho. MultIPAs: Applying Program Transformations to Introductory Programming Assignments for Data Augmentation. In ESEC/FSE 2022.

[5] Umair Z. Ahmed, Zhiyu Fan, Jooyong Yi, Omar I. Al-Bataineh, and Abhik Roychoudhury, Verifix: Verified repair of programming assignments. In TOSEM 2022.

[6] S. Gulwani, I. Radicek, and F. Zuleger. Automated clustering and program repair for introductory programming assignments. In PLDI 2018.

Acknowledgments

This work was supported by Portuguese national funds through FCT under projects UIDB/50021/2020, PTDC/CCI-COM/2156/2021, 2022.03537.PTDC and grant SFRH/BD/07724/2020. This work was also supported by European funds through COST Action CA2011; by the European Regional Development Fund under the Czech project AI\&Reasoning no.~CZ.02.1.01/0.0/0.0/15_003/0000466 (JP), Amazon Research Awards (JP), and by the Ministry of Education, Youth, and Sports within the program ERC CZ under the project POSTMAN no. LL1902. This article is part of the RICAIP project that has received funding from the EU's Horizon 2020 research and innovation program under grant agreement No 857306.