LLM-Driven Program Repair Using MaxSAT-based Fault Localization

Pedro Orvalho [1,2]

[1]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
[2]CIIRC, Czech Technical University in Prague, Czechia

QAVAS Work Meeting, University of Oxford

Wednesday 4th December, 2024

TÉCNICO LISBOA inesc id lisboa CZECH INSTITUTE OF INFORMATICS ROBOTICS AND CYBERNETICS CTU IN PRAGUE fct Fundação para a Ciência e a Tecnologia

Research Overview

- Program Synthesis:
  - Encodings for Enumeration-Based Program Synthesis. **CP 2019**;
  - SQUARES: A SQL Synthesizer Using Query Reverse Engineering. **VLDB 2020**;

Research Overview

- Program Synthesis:
  - Encodings for Enumeration-Based Program Synthesis. **CP 2019**;
  - SQUARES: A SQL Synthesizer Using Query Reverse Engineering. **VLDB 2020**;
- Maximum Satisfiability (MaxSAT):
  - UpMax: User partitioning for MaxSAT. **SAT 2023**;
  - AlloyMax: Bringing maximum satisfaction to relational specifications. **ESEC/FSE 2021**. [ACM SIGSOFT Distinguished Paper Award];

Research Overview

- Automated Program Repair
  - MultIPAs: Applying Program Transformations to Introductory Programming Assignments for Data Augmentation. **ESEC/FSE 2022**;
  - Graph Neural Networks For Mapping Variables Between Programs. **ECAI 2023**;
  - C-Pack of IPAs: A C90 Program Benchmark of Introductory Programming Assignments. **APR 2024**;
  - GitSEED: A Git-backed Automated Assessment Tool for Software Engineering and Programming Education. The 1st ACM **SIGCSE Virtual 2024**;
  - CFaults: Model-Based Diagnosis for Fault Localization in C with Multiple Test Cases. **FM 2024**;
  - Counterexample Guided Program Repair Using Zero-Shot Learning and MaxSAT-based Fault Localization. **[Under Review]**;
  - On Applying Invariant-Based Program Clustering to Introductory Programming Assignments. **[Under Review]**.

Research Overview

- Automated Program Repair
  - MultIPAs: Applying Program Transformations to Introductory Programming Assignments for Data Augmentation. **ESEC/FSE 2022**;
  - Graph Neural Networks For Mapping Variables Between Programs. **ECAI 2023**;
  - C-Pack of IPAs: A C90 Program Benchmark of Introductory Programming Assignments. **APR 2024**;
  - GitSEED: A Git-backed Automated Assessment Tool for Software Engineering and Programming Education. The 1st ACM **SIGCSE Virtual 2024**;
  - CFaults: Model-Based Diagnosis for Fault Localization in C with Multiple Test Cases. **FM 2024**;
  - Counterexample Guided Program Repair Using Zero-Shot Learning and MaxSAT-based Fault Localization. **[Under Review]**;
  - On Applying Invariant-Based Program Clustering to Introductory Programming Assignments. **[Under Review]**.

Automated Program Repair (APR)

APR - Motivation

- The increasing demand for programming education has given rise to all kinds of online evaluations focused on introductory programming assignments (IPAs):

APR - Motivation

- The increasing demand for programming education has given rise to all kinds of online evaluations focused on introductory programming assignments (IPAs):
  - MIT's MOOC, Introduction to CS, **reached 1.2 M enrollments** in 2018;

APR - Motivation

- The increasing demand for programming education has given rise to all kinds of online evaluations focused on introductory programming assignments (IPAs):
  - MIT's MOOC, Introduction to CS, **reached 1.2 M enrollments** in 2018;
  - In 2020, Stanford's CS MOOC had **more than 10K students**.

Automated Program Repair (APR)

Given a buggy program P_o and a set of input-output examples T (test suite).

Automated Program Repair (APR)

Given a buggy program P_o and a set of input-output examples T (test suite).

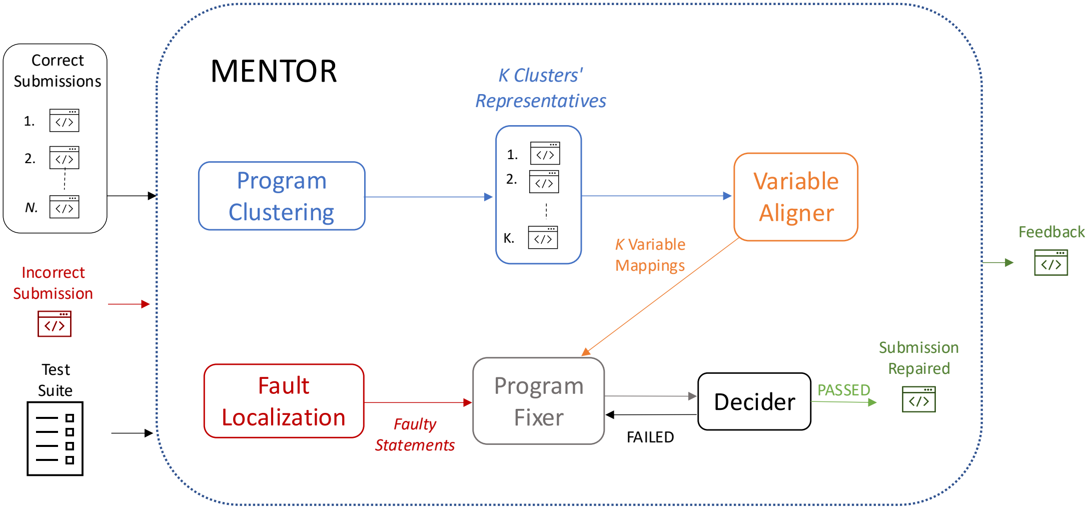
The goal of *Automated Program Repair* is to find a program P_f by **semantically change a subset S_1 of P_o's statements** ($S_1 \subseteq P_o$) for another set of statements S_2, s.t.,

$$P_f = ((P_o \setminus S_1) \cup S_2)$$

and

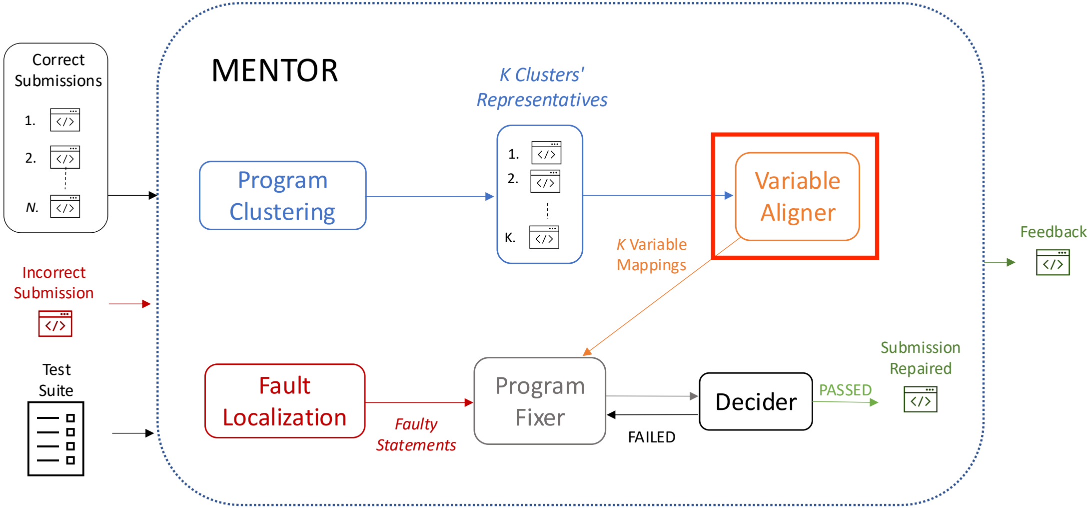
$$\forall \{t_{in}^i, t_{out}^i\} \in T \ : \ P_f(t_{in}^i) = t_{out}^i$$

MENTOR



Correct Submissions
1.
2.
N.
Incorrect Submission
Test Suite
MENTOR
Program Clustering
K Clusters' Representatives
1.
2.
K.
Variable Aligner
K Variable Mappings
Fault Localization
Faulty Statements
Program Fixer
Decider
FAILED
PASSED
Submission Repaired
Feedback

MENTOR



Correct
Submissions
1.
2.
N.
Incorrect
Submission
Test
Suite
MENTOR
Program
Clustering
K Clusters'
Representatives
1.
2.
K.
Variable
Aligner
K Variable
Mappings
Fault
Localization
Faulty
Statements
Program
Fixer
FAILED
Decider
PASSED
Submission
Repaired
Feedback

ECAI 23 - Graph Neural Networks For Mapping Variables Between Programs

Variable Mapping - Motivation

- Comparing two programs is **highly challenging**;

Variable Mapping - Motivation

- Comparing two programs is **highly challenging**;
- **A relation between** two programs' sets of **variables is required**;

Variable Mapping - Motivation

- Comparing two programs is **highly challenging**;
- **A relation between** two programs' sets of **variables is required**;
- Mapping variables between two programs is **useful for a variety of program related tasks**, such as, program equivalence, program repair, etc.

Variable Mapping - Motivation

1: Function that finds and returns the maximum number among `n1`, `n2` and `n3`.

```
int max(int n1, int n2, int n3)
{
  int m = n1 > n2 ? n1 : n2;
  return n3 > m ? n3 : m;
}
```

2: Function that finds and returns the maximum number among `x`, `y` and `z`.

```
int max(int x, int y, int z){
  int m = 0;
  m = x > m ? x : m;
  m = y > m ? y : m;
  return z > m ? z : m;
}
```

Variable Mapping - Motivation

3: Function that finds and returns the maximum number among `n1`, `n2` and `n3`.

```
int max(int n1, int n2, int n3)
{
   int m = n1 > n2 ? n1 : n2;
   return n3 > m ? n3 : m;
}
```

4: Function that finds and returns the maximum number among `x`, `y` and `z`.

```
int max(int x, int y, int z){
   int m = 0;
   m = x > m ? x : m;
   m = y > m ? y : m;
   return z > m ? z : m;
}
```

Variable Mapping: {m : m; n1 : x; n2 : y; n3 : z}.

Motivation

5: Function that finds and returns the maximum number among `n1`, `n2` and `n3`.

```
int max(int n1, int n2, int n3)
{
  int m = n1 > n2 ? n1 : n2;
  return n3 > m ? n3 : m;
}
```

6: Function that finds and returns the maximum number among `x`, `y` and `z`.

```
int max(int x, int y, int z){
  int m = 0;
  m = x > m ? x : m;
  m = y > m ? y : m;
  return z > m ? z : m;
}
```

Variable Mapping: {`m` : `m`; `n1` : `x`; `n2` : `y`; `n3` : `z`}.

Contribution

- A graph **program representation that takes advantage of the structural information of the *abstract syntax trees (ASTs)*** of programs;

Contribution

- A graph **program representation that takes advantage of the structural information of the *abstract syntax trees (ASTs)*** of programs;
- Our program representation is **agnostic to the names of the variables**;

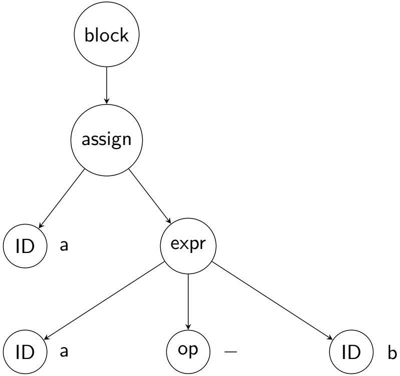
Contribution

- A graph **program representation that takes advantage of the structural information of the *abstract syntax trees (ASTs)*** of programs;
- Our program representation is **agnostic to the names of the variables**;
- **Map the variables between a correct program and a faulty one using *Graph Neural Networks (GNNs)***.

Program Representation

7: An expression that uses int variables `a` and `b`, previously declared in the program.

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



(a) Part of the AST representation.

Program Representation

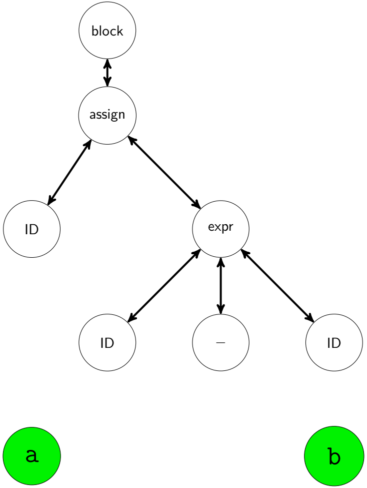
8: An expression that uses int variables `a` and `b`, previously declared in the program.

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

Types of edges:

AST ↔

Variable Node

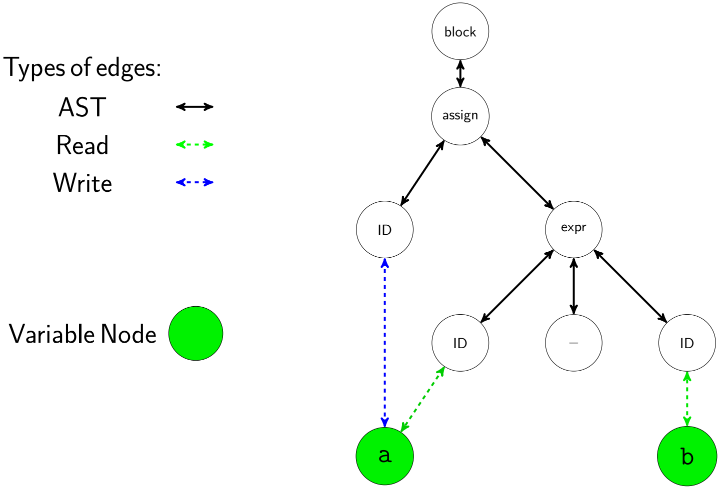


(b) Our program representation.

Program Representation

9: An expression that uses int variables a and b, previously declared in the program.

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

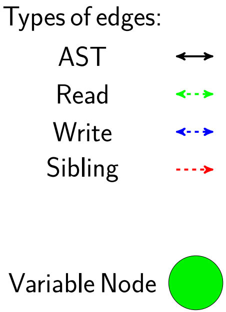


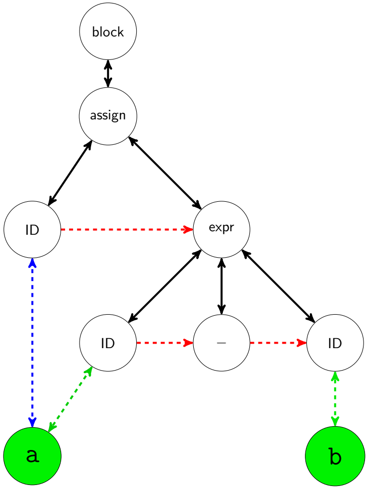
(c) Our program representation.

Program Representation

10: An expression that uses int variables a and b, previously declared in the program.

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



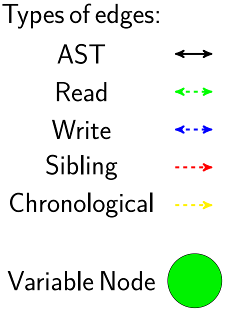


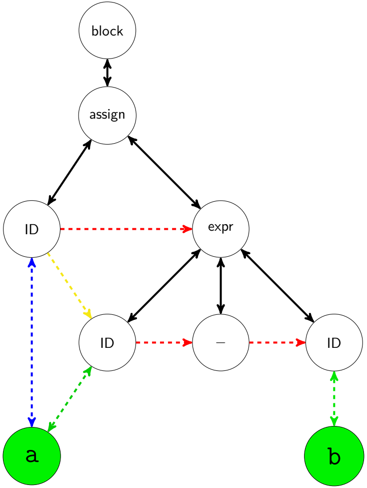
(d) Our program representation.

Program Representation

11: An expression that uses int variables `a` and `b`, previously declared in the program.

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





(e) Our program representation.

Graph Neural Networks (GNNs)

- We use a **Relational Graph Convolutional Neural Network (RGCN)**;

Graph Neural Networks (GNNs)

- We use a **Relational Graph Convolutional Neural Network (RGCN)**;
- We perform ***message passing*** **between the nodes of our representations**, so that information can be passed between the local constituents;

Graph Neural Networks (GNNs)

- We use a **Relational Graph Convolutional Neural Network (RGCN)**;
- We perform ***message passing*** **between the nodes of our representations**, so that information can be passed between the local constituents;
- After several message passing rounds, we obtain **numerical vectors corresponding to each variable node** in the two programs;

Graph Neural Networks (GNNs)

- We use a **Relational Graph Convolutional Neural Network (RGCN)**;
- We perform ***message passing* between the nodes of our representations**, so that information can be passed between the local constituents;
- After several message passing rounds, we obtain **numerical vectors corresponding to each variable node** in the two programs;
- We compute **scalar products between each possible combination of variable nodes** in the two programs, followed by a softmax function.

Data Augmentation

- We use C-Pack-IPAs [Orvalho et al., 2022], a set of 10 **introductory programming assignments**, comprising **486 faulty programs**;

Data Augmentation

- We use C-PACK-IPAS [Orvalho et al., 2022], a set of 10 **introductory programming assignments**, comprising **486 faulty programs**;
- Since we need to know the real variable mappings between programs to evaluate our representation, we used **MultIPAs [Orvalho et al., 2022] to generate a dataset of pairs of correct/incorrect programs**:

Data Augmentation

- We use C-Pack-IPAs [Orvalho et al., 2022], a set of 10 **introductory programming assignments**, comprising **486 faulty programs**;
- Since we need to know the real variable mappings between programs to evaluate our representation, we used **MultIPAs [Orvalho et al., 2022] to generate a dataset of pairs of correct/incorrect programs**:
  - MultIPAs can perform six syntactic program mutations;

Data Augmentation

- We use C-PACK-IPAS [Orvalho et al., 2022], a set of 10 **introductory programming assignments**, comprising **486 faulty programs**;
- Since we need to know the real variable mappings between programs to evaluate our representation, we used **MultIPAs [Orvalho et al., 2022] to generate a dataset of pairs of correct/incorrect programs**:
  - MULTIPAS can perform six syntactic program mutations;
  - MULTIPAS can introduce three kinds of bugs: wrong comparison operator (WCO), variable misuse (VM), and missing expression (ME).

Variable Mapping - Results

	Buggy Programs (Total = 186366)
Correct Mappings	179470 (96.49%)

Table 1: Validation Performance after 20 training epochs.

Variable Mapping - Results

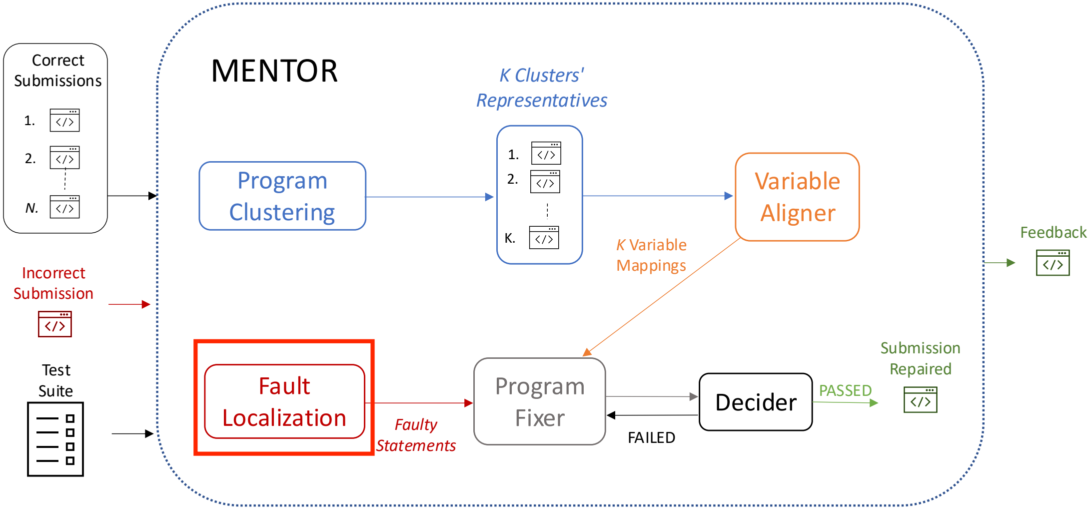
	Buggy Programs (Total = 186366)
Correct Mappings	179470 (96.49%)

Table 1: Validation Performance after 20 training epochs.

Evaluation Metric	Buggy Programs
# Correct Mappings	82.77%
Avg Overlap Coefficient	95.05%

Table 2: Test Performance.

MENTOR



Correct Submissions
1.
2.
N.
Incorrect Submission
Test Suite
MENTOR
Program Clustering
K Clusters' Representatives
1.
2.
K.
Variable Aligner
K Variable Mappings
Fault Localization
Faulty Statements
Program Fixer
Decider
PASSED
FAILED
Submission Repaired
Feedback

FM24 - CFaults: Model-Based Diagnosis for Fault Localization in C with Multiple Test Cases

Fault Localization - Motivation

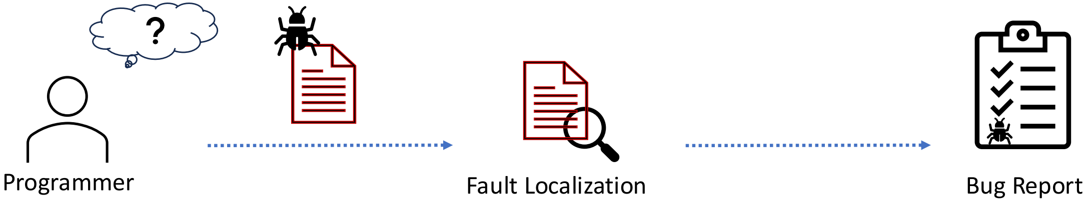
- Debugging is one of the most time-consuming and expensive tasks in software development.

Fault Localization - Motivation

- Debugging is one of the most time-consuming and expensive tasks in software development.
- In 2024, the estimated global cost of Crowdstrike's error that hit Microsoft systems, is 5.4 Billion US$ [The Guardian UK, 2024].

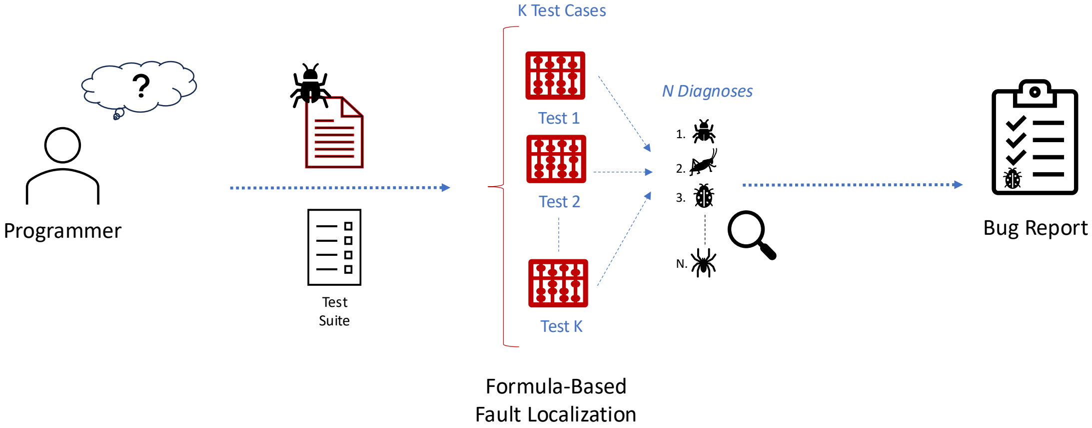
Fault Localization

- Given a buggy program, *fault localization (FL)* involves identifying locations in the program that could cause a faulty behaviour (bug).



Formula-Based Fault Localization (FBFL)

- FBFL methods encode the localization problem into **several optimization problems** to identify a minimal set of bugs (diagnoses).



Formula-Based Fault Localization (FBFL)

Current Limitations

FBFL tools especially for programs with multiple faults:

Formula-Based Fault Localization (FBFL)

Current Limitations

FBFL tools especially for programs with multiple faults:

- **do not ensure a minimal diagnosis** across all failing tests (e.g., BugAssist);

Formula-Based Fault Localization (FBFL)

Current Limitations

FBFL tools especially for programs with multiple faults:

- **do not ensure a minimal diagnosis** across all failing tests (e.g., BugAssist);
- may produce an overwhelming number of **redundant sets of diagnoses** (e.g., SNIPER).

Contribution

- We formulate the FL problem as a **single optimization problem**;

Contribution

- We formulate the FL problem as a **single optimization problem**;
- We leverage MaxSAT and the theory of *Model-Based Diagnosis (MBD)* [Reiter et al., 1987, Ignatiev et al., 2019], **integrating all failing test cases simultaneously**;

Contribution

- We formulate the FL problem as a **single optimization problem**;
- We leverage MaxSAT and the theory of *Model-Based Diagnosis (MBD)* [Reiter et al., 1987, Ignatiev et al., 2019], **integrating all failing test cases simultaneously**;
- We implement this MBD approach in a publicly available tool called CFaults.

Partial Maximum Satisfiability (MaxSAT)

Hard:	$h_1 : (v_1 \vee v_2)$	$h_2 : (\neg v_2 \vee v_3)$	$h_3 : (\neg v_1 \vee \neg v_3)$	$h_4 : (v_4 \vee v_5)$
	$h_5 : (\neg v_5 \vee v_6)$	$h_6 : (\neg v_4 \vee \neg v_6)$	$h_7 : (\neg v_3 \vee \neg v_6)$	
Soft:	$s_1 : (\neg v_1)$	$s_2 : (\neg v_3)$	$s_3 : (\neg v_4)$	$s_4 : (\neg v_6)$

Figure 1: Example of a partial MaxSAT formula.

Model-Based Diagnosis

- A system description $\mathcal{P}$ **is composed of a set of components** $\mathcal{C} = \{c_1, \ldots, c_n\}$.

Model-Based Diagnosis

- A system description $\mathcal{P}$ **is composed of a set of components** $\mathcal{C} = \{c_1, \ldots, c_n\}$.
- Each component in $\mathcal{C}$ can be declared **healthy** or **unhealthy**.

Model-Based Diagnosis

- A system description $\mathcal{P}$ **is composed of a set of components** $\mathcal{C} = \{c_1, \ldots, c_n\}$.
- Each component in $\mathcal{C}$ can be declared **healthy** or **unhealthy**.
- For each component $c \in \mathcal{C}$, $h(c) = 0$ **if** c **is unhealthy, otherwise,** $h(c) = 1$.

Model-Based Diagnosis

- A system description $\mathcal{P}$ **is composed of a set of components** $\mathcal{C} = \{c_1, \ldots, c_n\}$.
- Each component in $\mathcal{C}$ can be declared **healthy** or **unhealthy**.
- For each component $c \in \mathcal{C}$, $h(c) = 0$ **if** c **is unhealthy, otherwise,** $h(c) = 1$.
- $\mathcal{P}$ is described by a CNF formula, where $\mathcal{F}_c$ denotes the encoding of component c:

$$\mathcal{P} \triangleq \bigwedge\nolimits_{c \in \mathcal{C}} (\neg h(c) \vee \mathcal{F}_c) \tag{1}$$

Model-Based Diagnosis

- **Observations represent deviations** from the expected system behaviour.

Model-Based Diagnosis

- **Observations represent deviations** from the expected system behaviour.
- An observation, denoted as o, can be encoded in CNF as a set of unit clauses.

Model-Based Diagnosis

- **Observations represent deviations** from the expected system behaviour.
- An observation, denoted as o, can be encoded in CNF as a set of unit clauses.
- In our work, **the failing test cases represent the set of observations**.

Model-Based Diagnosis

- **Observations represent deviations** from the expected system behaviour.
- An observation, denoted as o, can be encoded in CNF as a set of unit clauses.
- In our work, **the failing test cases represent the set of observations**.
- A system $\mathcal{P}$ is considered **faulty if there exists an inconsistency with a given observation o when all components are declared healthy**:

$$\mathcal{P} \wedge o \wedge \bigwedge_{c \in \mathcal{C}} h(c) \models \bot \tag{2}$$

Model-Based Diagnosis

- The problem of model-based diagnosis (MBD) aims to **identify a set of components which, if declared unhealthy, restore consistency**;

Model-Based Diagnosis

- The problem of model-based diagnosis (MBD) aims to **identify a set of components which, if declared unhealthy, restore consistency**;
- For a given MBD problem $\langle \mathcal{P}, \mathcal{C}, o \rangle$, a set of system components $\Delta \subseteq \mathcal{C}$ is a diagnosis iff:

$$\mathcal{P} \wedge o \wedge \bigwedge_{c \in \mathcal{C} \setminus \Delta} h(c) \wedge \bigwedge_{c \in \Delta} \neg h(c) \nvDash \bot \tag{3}$$

Model-Based Diagnosis

- The problem of model-based diagnosis (MBD) aims to **identify a set of components which, if declared unhealthy, restore consistency**;
- For a given MBD problem $\langle \mathcal{P}, \mathcal{C}, o \rangle$, a set of system components $\Delta \subseteq \mathcal{C}$ is a diagnosis iff:

$$\mathcal{P} \wedge o \wedge \bigwedge_{c \in \mathcal{C} \setminus \Delta} h(c) \wedge \bigwedge_{c \in \Delta} \neg h(c) \nvDash \bot \quad (3)$$

- A diagnosis Δ is:

Model-Based Diagnosis

- The problem of model-based diagnosis (MBD) aims to **identify a set of components which, if declared unhealthy, restore consistency**;
- For a given MBD problem $\langle \mathcal{P}, \mathcal{C}, o \rangle$, a set of system components $\Delta \subseteq \mathcal{C}$ is a diagnosis iff:

$$\mathcal{P} \wedge o \wedge \bigwedge_{c \in \mathcal{C} \setminus \Delta} h(c) \wedge \bigwedge_{c \in \Delta} \neg h(c) \not\models \bot \quad (3)$$

- A diagnosis Δ is:
    - **minimal** iff no subset of Δ, $\Delta' \subsetneq \Delta$, is a diagnosis;

Model-Based Diagnosis

- The problem of model-based diagnosis (MBD) aims to **identify a set of components which, if declared unhealthy, restore consistency**;
- For a given MBD problem $\langle \mathcal{P}, \mathcal{C}, o \rangle$, a set of system components $\Delta \subseteq \mathcal{C}$ is a diagnosis iff:

$$\mathcal{P} \land o \land \bigwedge_{c \in \mathcal{C} \setminus \Delta} h(c) \land \bigwedge_{c \in \Delta} \neg h(c) \nvDash \bot \tag{3}$$

- A diagnosis Δ is:
  - **minimal** iff no subset of Δ, $\Delta' \subsetneq \Delta$, is a diagnosis;
  - Δ is of **minimal cardinality** if there is no other diagnosis $\Delta'' \subseteq \mathcal{C}$ with $|\Delta''| < |\Delta|$;

Model-Based Diagnosis

- The problem of model-based diagnosis (MBD) aims to **identify a set of components which, if declared unhealthy, restore consistency**;
- For a given MBD problem $\langle \mathcal{P}, \mathcal{C}, o \rangle$, a set of system components $\Delta \subseteq \mathcal{C}$ is a diagnosis iff:

$$\mathcal{P} \wedge o \wedge \bigwedge_{c \in \mathcal{C} \setminus \Delta} h(c) \wedge \bigwedge_{c \in \Delta} \neg h(c) \nvDash \bot \tag{3}$$

- A diagnosis Δ is:
  - **minimal** iff no subset of Δ, $\Delta' \subsetneq \Delta$, is a diagnosis;
  - Δ is of **minimal cardinality** if there is no other diagnosis $\Delta'' \subseteq \mathcal{C}$ with $|\Delta''| < |\Delta|$;
  - is **redundant** if it is not subset-minimal [Ignatiev et al., 2019].

Model-Based Diagnosis

To encode the MBD problem with one observation with partial MaxSAT:

- The set of **clauses that encode $\mathcal{P}$ represents the set of hard clauses**;

Model-Based Diagnosis

To encode the MBD problem with one observation with partial MaxSAT:

- The set of **clauses that encode $\mathcal{P}$ represents the set of hard clauses**;
- The soft clauses consists of unit clauses that aim to **maximize the set of healthy components**, i.e.,:

$$\bigwedge_{c \in \mathcal{C}} h(c);$$

Model-Based Diagnosis

To encode the MBD problem with one observation with partial MaxSAT:

- The set of **clauses that encode $\mathcal{P}$ represents the set of hard clauses**;
- The soft clauses consists of unit clauses that aim to **maximize the set of healthy components**, i.e.,:

$$\bigwedge_{c \in \mathcal{C}} h(c);$$

- This encoding enables enumerating subset **minimal diagnoses, considering a single observation**;

Model-Based Diagnosis with Multiple Test Cases

We **integrate all failing test cases** in a single MaxSAT formula.

Model-Based Diagnosis with Multiple Test Cases

We **integrate all failing test cases** in a single MaxSAT formula.

- We **generate only minimal diagnoses** capable of identifying all faulty components within the system, in our case, a C program;

Model-Based Diagnosis with Multiple Test Cases

We **integrate all failing test cases** in a single MaxSAT formula.

- We **generate only minimal diagnoses** capable of identifying all faulty components within the system, in our case, a C program;
- Given m observations, $\mathcal{O} = \{o_1, \ldots, o_m\}$, a distinct replica of the system, denoted as $\mathcal{P}_i$, is required for each observation o_i;

Model-Based Diagnosis with Multiple Test Cases

We **integrate all failing test cases** in a single MaxSAT formula.

- We **generate only minimal diagnoses** capable of identifying all faulty components within the system, in our case, a C program;
- Given m observations, $\mathcal{O} = \{o_1, \ldots, o_m\}$, a distinct replica of the system, denoted as $\mathcal{P}_i$, is required for each observation o_i;
- The hard clauses, ϕ_h, in our MaxSAT formulation correspond to:

$$\phi_h = \bigwedge_{o_i \in \mathcal{O}} (\mathcal{P}_i \wedge o_i);$$

Model-Based Diagnosis with Multiple Test Cases

We **integrate all failing test cases** in a single MaxSAT formula.

- We **generate only minimal diagnoses** capable of identifying all faulty components within the system, in our case, a C program;
- Given m observations, $\mathcal{O} = \{o_1, \ldots, o_m\}$, a distinct replica of the system, denoted as $\mathcal{P}_i$, is required for each observation o_i;
- The hard clauses, ϕ_h, in our MaxSAT formulation correspond to:

$$\phi_h = \bigwedge_{o_i \in \mathcal{O}} (\mathcal{P}_i \wedge o_i);$$

- The soft clauses are formulated as:

$$\phi_s = \bigwedge_{c \in \mathcal{C}} h(c).$$

Model-Based Diagnosis with Multiple Test Cases

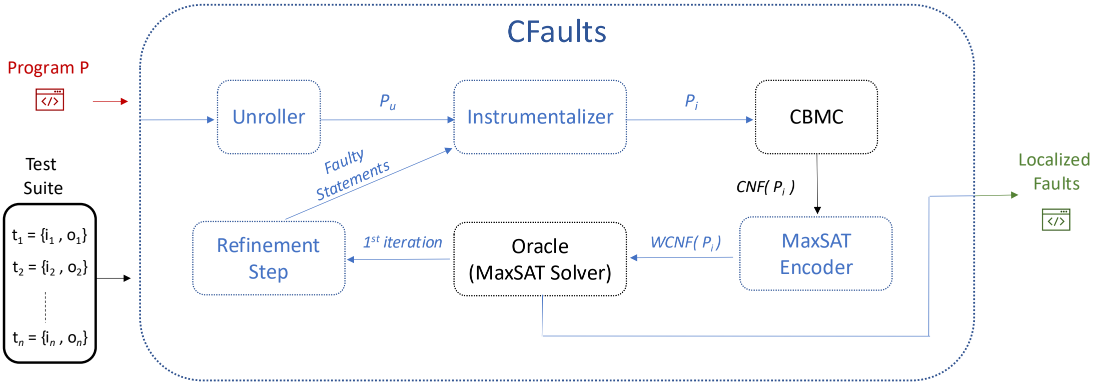
- Given a MaxSAT solution, **the set of unhealthy components ($h(c) = 0$), corresponds to a subset-minimal aggregated diagnosis**.

Model-Based Diagnosis with Multiple Test Cases

- Given a MaxSAT solution, **the set of unhealthy components ($h(c) = 0$), corresponds to a subset-minimal aggregated diagnosis**.
- This diagnosis makes the system **consistent with all observations**, as follows:

$$\bigwedge_{o_i \in \mathcal{O}} (\mathcal{P}_i \wedge o_i) \wedge \bigwedge_{c \in \mathcal{C} \setminus \Delta} h(c) \wedge \bigwedge_{c \in \Delta} \neg h(c) \nvDash \bot \tag{4}$$

CFaults



CFaults
Program P
Unroller
P_u
Instrumentalizer
P_i
CBMC
Faulty Statements
CNF(P_i)
Localized Faults
Test Suite
$t_1 = \{i_1, o_1\}$
$t_2 = \{i_2, o_2\}$
$t_n = \{i_n, o_n\}$
Refinement Step
1st iteration
Oracle (MaxSAT Solver)
WCNF(P_i)
MaxSAT Encoder

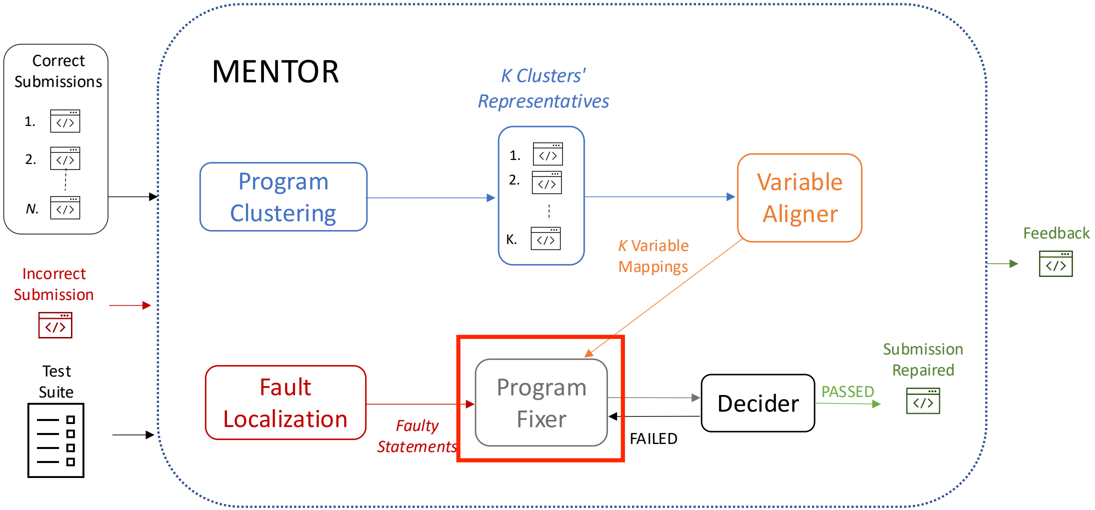
CFaults- Results

Benchmark: **C-Pack-IPAs**

	Valid Diagnosis	**Memouts**	**Timeouts**
BugAssist	454 (93.42%)	0 (0.0%)	32 (6.58%)
SNIPER	446 (91.77%)	4 (0.82%)	36 (7.41%)
CFaults	**483 (99.38%)**	1 (0.21%)	2 (0.41%)

Table 3: BugAssist, SNIPER and CFaults fault localization results on C-Pack-IPAs.

MENTOR



Correct Submissions
1.
2.
N.
Incorrect Submission
Test Suite
MENTOR
Program Clustering
K Clusters' Representatives
1.
2.
K.
Variable Aligner
K Variable Mappings
Fault Localization
Faulty Statements
Program Fixer
Decider
PASSED
FAILED
Submission Repaired
Feedback

Counterexample Guided APR Using MaxSAT-based Fault Localization

Motivation

12: Semantically incorrect program. Faults: {4,8}.

```
1  int main(){ //finds max of 3 nums
2      int f,s,t;
3      scanf("%d%d%d",&f,&s,&t);
4      if (f < s && f >= t)
5          printf("%d",f);
6      else if (s > f && s >= t)
7          printf("%d",s);
8      else if (t < f && t < s)
9          printf("%d",t);
10
11     return 0;
12 }
```

Motivation

13: Semantically incorrect program. Faults: {4,8}.

```
1  int main(){ //finds max of 3 nums
2      int f,s,t;
3      scanf("%d%d%d",&f,&s,&t);
4      if (f < s && f >= t)
5          printf("%d",f);
6      else if (s > f && s >= t)
7          printf("%d",s);
8      else if (t < f && t < s)
9          printf("%d",t);
10
11     return 0;
12 }
```

LLMs for code (LLMCs)

- GRANITE and CODEGEMMA **cannot fix** the buggy program within 90 secs;

Motivation

14: Semantically incorrect program. Faults: {4,8}.

```
1  int main(){ //finds max of 3 nums
2      int f,s,t;
3      scanf("%d%d%d",&f,&s,&t);
4      if (f < s && f >= t)
5          printf("%d",f);
6      else if (s > f && s >= t)
7          printf("%d",s);
8      else if (t < f && t < s)
9          printf("%d",t);
10
11     return 0;
12 }
```

LLMs for code (LLMCs)

- GRANITE and CODEGEMMA **cannot fix** the buggy program within 90 secs;
- Even if we provide the assignment's **description and IO tests**.

Program Sketches

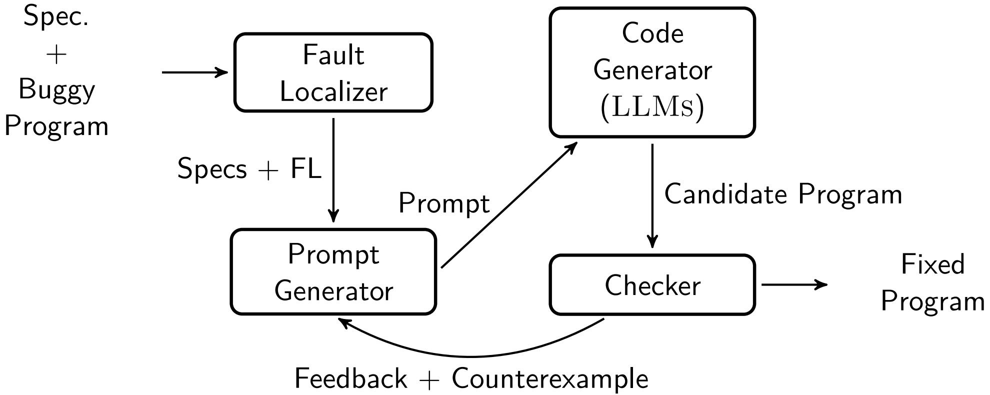
15: Semantically incorrect program. Faults :{4,8}.

```
1  int main(){ //finds max of 3 nums
2      int f,s,t;
3      scanf("%d%d%d",&f,&s,&t);
4      if (f < s && f >= t)
5          printf("%d",f);
6      else if (s > f && s >= t)
7          printf("%d",s);
8      else if (t < f && t < s)
9          printf("%d",t);
10
11     return 0;
12 }
```

16: Program sketch with holes.

```
1  int main(){
2      int f,s,t;
3      scanf("%d%d%d",&f,&s,&t);
4      @ HOLE 1 @
5          printf("%d",f);
6      else if (s > f && s >= t)
7          printf("%d",s);
8      @ HOLE 2 @
9          printf("%d",t);
10
11     return 0;
12 }
```

Counterexample Guided Automated Repair



Prompt Example

~~~~
Fix all semantic bugs in the buggy program
below. Modify the code as little as possible.
Do not provide any explanation.

### Problem Description ###
Write a program that determines and
prints the largest of three integers
given by the user.

### Test Suite
#input:
6 2 1
#output:
6
// The other input-output tests

# Reference Implementation
(Do not copy this program) <c> #
```c
int main(){
  // Reference Implementation
}
```

### Buggy Program <c> ###
```c
int main(){
  // Buggy program from Listing 1
}
```

### Fixed Program <c> ###
```c
~~~~

LLM-Driven APR with CFaults

	Prompt Configurations			
LLMs	**De-TS**	**De-TS-CE**	**Sk_De-TS**	**Sk_De-TS-CE**
CodeGemma	597 (41.7%)	606 (42.3%)	682 (47.7%)	**688 (48.1%)**
CodeLlama	492 (34.4%)	500 (34.9%)	**573 (40.0%)**	561 (39.2%)
Gemma	496 (34.7%)	492 (34.4%)	532 (37.2%)	**534 (37.3%)**
Granite	626 (43.7%)	624 (43.6%)	**691 (48.3%)**	681 (47.6%)
Llama3	564 (39.4%)	590 (41.2%)	578 (40.4%)	**591 (41.3%)**
Phi3	494 (34.5%)	489 (34.2%)	**547 (38.2%)**	535 (37.4%)
Verifix	90 (6.3%)			
Clara	495 (34.6%)			

Table 4: The number of programs fixed by each LLM under various configurations. Mapping abbreviations to configuration names: **De** - IPA *Description*, **TS** - *Test Suite*, **CE** - *Counterexample*, **SK** - *Sketches*.

LLM-Driven APR with CFaults + VMs

	Prompt configurations with access to Reference Implementations and Variable Mappings			
LLMs	**Sk_De-TS**	**Sk_De-TS-CE**	**Sk_De-TS-CE-CPA-VM**	**Sk_De-TS-CE-RI-VM**
CodeGemma	682 (47.7%)	688 (48.1%)	**782 (54.6%)**	780 (54.5%)
CodeLlama	573 (40.0%)	561 (39.2%)	**681 (47.6%)**	677 (47.3%)
Gemma	532 (37.2%)	534 (37.3%)	756 (52.8%)	**766 (53.5%)**
Granite	691 (48.3%)	681 (47.6%)	901 (63.0%)	**921 (64.4%)**
Llama3	578 (40.4%)	591 (41.3%)	**792 (55.3%)**	720 (50.3%)
Phi3	547 (38.2%)	535 (37.4%)	**691 (48.3%)**	**691 (48.3%)**

Table 5: The number of programs fixed by each LLM under various configurations. Mapping abbreviations to configuration names: **CPA** - *Closest Program using* AASTs, **De** - IPA *Description*, **RI** - *Reference Implementation*, **SK** - *Sketches*, **TS** - *Test Suite*, **VM** - *Variable Mapping*.

How Can I Collaborate with You?

I have gained extensive experience in:

- symbolic methods including:
  - Constraint Solving (MaxSAT, SAT, SMT);
  - Program Verification;
  - Model-Based Diagnosis;
  - Program Synthesis and Repair.

How Can I Collaborate with You?

I have gained extensive experience in:

- symbolic methods including:
  - Constraint Solving (MaxSAT, SAT, SMT);
  - Program Verification;
  - Model-Based Diagnosis;
  - Program Synthesis and Repair.
- developing software and experimental tools;

How Can I Collaborate with You?

I have gained extensive experience in:

- symbolic methods including:
  - Constraint Solving (MaxSAT, SAT, SMT);
  - Program Verification;
  - Model-Based Diagnosis;
  - Program Synthesis and Repair.
- developing software and experimental tools;
- hosting and running LLMs for chat-based procedures.

Thank you!

https://pmorvalho.github.io

References

Reiter, Raymond (1987)
A Theory of Diagnosis from First Principles.
Artif. Intell. 1987.

Do, Hyunsook and Elbaum, Sebastian G. and Rothermel, Gregg (2005)
Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact.
Empir. Softw. Eng. 2005.

Manu Jose and Rupak Majumdar (2011)
Cause clue clauses: error localization using maximum satisfiability.
PLDI 2011.

Lamraoui, Si-Mohamed and Nakajima, Shin (2016)
A Formula-based Approach for Automatic Fault Localization of Multi-fault Programs.
J. Inf. Process. 24(1), 88 – 98.

References

Ignatiev, Alexey and Morgado, António and Weissenbacher, Georg and Marques-Silva, João (2019)
Model-Based Diagnosis with Multiple Observations.
IJCAI 2019.

Orvalho, P. and Janota, M. and Manquinho, V. (2022)
C-Pack of IPAs: A C90 Program Benchmark of Introductory Programming Assignments.
arXiv:2206.08768.

The Guardian - Year 2000 Problem
https://www.theguardian.com/commentisfree/2019/dec/31/millennium-bug-face-fears-y2k-it-systems
The Guardian 2019.

The Guardian UK - Crowdstrike Meltdown
https://www.theguardian.com/technology/article/2024/jul/24/crowdstrike-outage-companies-cost.
The Guardian UK.

References

Ahmed, Umair Z and Fan, Zhiyu and Yi, Jooyong and Al-Bataineh, Omar I and Roychoudhury, Abhik (2022)

Verifix: Verified repair of programming assignments.

TOSEM 22 12(3), 45 – 678.

Orvalho, Pedro and Janota, Mikoláš and Manquinho, Vasco (2022)

MultIPAs: Applying Program Transformations to Introductory Programming Assignments for Data Augmentation.

ESEC/FSE 2022.

Gulwani, Sumit and Radiček, Ivan and Zuleger, Florian (2018)

Automated clustering and program repair for introductory programming assignments.

PLDI 18 52(4), 465 – 480.