PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - I

A Thesis Submitted

In Partial Fulfillment of the Requirements

For the Degree of M.Tech

by

Ayush Sahni

21111019

to the

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

May 2024

Page intentionally left blank

Declaration

This is to certify that the thesis titled **"PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - I"** has been authored by me. It presents the research conducted by me under the supervision of **PROF. PURUSHOTTAM KAR** and **PROF. AMEY KARKARE**.

To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Further, due credit has been attributed to the relevant state-of-the-art and collaborations (if any) with appropriate citations and acknowledgments, in line with established norms and practices.

Ayush Sahni

Name: Ayush Sahni (21111019)
Program: M.Tech
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur, Kanpur, 208016.
May 2024

Page intentionally left blank

Declaration (To be submitted at DOAA Office)

I hereby declare that

1. The research work presented in the thesis titled **"PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - I"** has been conducted by me under the guidance by my supervisor(s) **PROF. PURUSHOTTAM KAR** and **PROF. AMEY KARKARE**.

2. The thesis has been formatted as per Institute guidelines.

3. The content of the thesis (text, illustration, data, plots, pictures etc.) is original and is the outcome of my research work. Any relevant material taken from the open literature has been referred and cited, as per established ethical norms and practices.

4. All collaborations and critiques that have contributed to giving the thesis its final shape have been duly acknowledged and credited.

5. Care has been taken to give due credit to the state-of-the-art in the thesis research area.

6. I fully understand that in case the thesis is found to be unoriginal or plagiarized, the Institute reserves the right to withdraw the thesis from its archive and also revoke the associated degree conferred. Additionally, the Institute also reserves the right to apprise all concerned sections of society of the matter, for their information and necessary action (if any).

Ayush Sahni

Name: Ayush Sahni
Program: M.Tech
Department of Computer Science and Engineering
Roll No.: 21111019
Indian Institute of Technology Kanpur, Kanpur, 208016.
May 2024

Page intentionally left blank

Certificate

It is certified that the work contained in the thesis titled **"PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - I"** by **AYUSH SAHNI** has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

KAR Pu Amey Karkare

Prof. Purushottam Kar, Prof. Amey Karkare
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur, 208016.
May 2024

Page intentionally left blank

Abstract

Name of student: **Ayush Sahni** Roll no: **21111019**
Degree for which submitted: **M.Tech**
Department: **Department of Computer Science and Engineering**
Thesis title: **PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - I**

Name of thesis supervisor: **Prof. Purushottam Kar**, **Prof. Amey Karkare**
Month and year of thesis submission: **May 2024**

As [8][3][6] have discussed, for beginner programmers, the compilation error given by compiler seems puzzling and is difficult to understand. The compilation error messages are technically correct but they usually fail to effectively help a beginner programmer on how to address their errors. In many cases, this is true for experienced programmers also but for someone who has recently started programming, it requires too much effort from their side as well as from their mentors. With large number of students, it becomes more difficult for mentors to give sufficient attention to each student.

Hence, the use of automatic program repair and suggestion to fix compilation errors is very much needed. This field is currently a focus of research. Such tool will be of great convenience not just to beginners but to experienced programmers as well.

We have built upon MACER++ [8] and we have modified its repair class structure and repair application algorithm. We have introduced two new operations viz. replace and move in MACER++'s repair class. Our approach gives nearly 70.5% pred@5 [3][8] and 72% pred@10 measure on TRACER's [1] single line test dataset.

We have also added event logging feature to PRIORITY [8] [6] using which we can further improve ML algorithms that can be used to label new problems more accurately. For more details on this, please refer [9]

Page intentionally left blank

Acknowledgments

I would like to thank my thesis supervisors, Prof. Purushottam Kar and Prof. Amey Karkare, for all the help and advice they have given me. Without their guidance and support, I would not have been able to finish this thesis.
I would also like to thank my teammates Utkarsh Srivastava and Jeet Sarangi for their help. I have had lot of discussions and exchange of ideas with them. Additionally, I would also like to extend my acknowledgement to my seniors, Debanjan Chatterjee, Preeti Singh, Mayank Bansal, Fahad Shaikh and Sharath HP, for their guidance and support.
I would also like to thank CSE department instructors for their guidance and instruction throughout my time at IIT Kanpur. The knowledge I gained here has helped me grow both professionally and personally.
In the end, I want to thank my parents for their unwavering faith in me and encouragement throughout my life.
This thesis was compiled using a template graciously made available by Olivier Commowick `http://olivier.commowick.org/thesis_template.php`. The template was suitably modified to adapt to the requirements of the Indian Institute of Technology Kanpur.

Ayush Sahni
May 2024

Contents

List of Figures

List of Tables

Page intentionally left blank

CHAPTER 1

Introduction

Contents

As discussed in [8][3][1], computer programmers primarily rely on feedback from programming environments like compilers to modify their code. Compiler error messages are accurate in theory but they don't help new programmers much. Such inexperienced programmers may become confused by the terse language used in compiler error messages, which may contain words like "prototype" or "dereference". Usually, compilers only concentrate on technical aspects and throw errors in such a way which does not correspond with the user's basic error pattern. There are several examples that show compiler error messages can be somewhat frustating. Inexperienced programmers frequently turn to human mentors for helping and correcting their code. This method is not scalable considering the growing number of students as compared to mentors. Because of its potential in programming instruction, automatic programme repair tools need to be developed which can benifit programmers as well as educators.

In recent work [3], an AI-based pipeline has been developed that takes an incorrect program as input and as output, gives compilation-error free repaired program. Compared to other techniques like [5], this methodology is better suited for educational applications since it does not only directly gives a repaired program which compiles anyhow but the repairs suggested by it resemble close to what student wrote. This helps them learn to correct their mistakes on their own rather than directly getting a repaired program which would undermine learning.

1.1 Our Contributions

In attempt to apply repair to an erroneous source line, MACER and MACER++ consider only two types of operations viz. insert and delete. Taking it one step further, we have introduced 2 new operations viz. replace and move. So, in our work, a symbol can be either inserted, deleted, replaced or moved to repair the erroneous source line. This also makes the type of repair more understandle by beginner programmers. For example: Suppose we want to suggest students to help them repair their mistake. Consider the source line which has errors:
if (a = 1 || a = 21 || a = 31)
Our application, similar to MACER and MACER++, but in more general cases as well, will tell correctly them to replace all the three instances of '=' with '=='.

Consider the erroneous source line written by student:
printf ("hello world" ;).

Note that the symbol ';' needs to be moved one index towards right to fix the error. In case of MACER and MACER++, the predicted repair class for this source line will tell to delete ';' and insert ';'. Our proposed method is intended to tell them to move ';' which is easier to understand.

This thesis is one of the three associate theses that were completed together. For complete context, please refer to [7] and [9]

CHAPTER 2

Related Works

Contents

2.1 TRACER

As discussed in [1], with a special combination of deep learning and programming language theory tools, TRACER[1] produces repairs that not only guarantee successful compilation but also closely match fixes performed by students who have encountered similar mistakes. TRACER is more effective at providing real-time feedback during lab or tutorial sessions because of its targeted approach, which offers particular corrections instead of merely making sure code compiles. This sets it apart from other solutions that only concentrate on reaching a given compilation success rate.

There have been some work for creating specialised compilers for inexperienced users with the goal of improving error correction over normal compilers and offering superior feedback. This requires lot of work to be put in by compiler designers and they must do it again for every compiler that is used. TRACER acknowledges that the goal of program repair in educational settings goes beyond just getting rid of compilation faults by any means. For example, by eliminating the error line completely.
Rather than hiding the underlying mistakes from students, TRACER intends to provide them the tools they need to solve similar problems on their own in the future. MACER[3] has built upon this idea of TRACER.

To repair code, TRACER uses 3 modular steps: Error localization(locate lines where repair must be performed), abstract code repair and concretization.
For the error localization phase, TRACER leveraged an important observation from a dataset of introductory C programs, where students made single-line edits. They noted that if a compiler error message attributes the compilation error to line number l, it is beneficial to consider lines l-1, l and l+1 as potential candidates for repair.
This approach achieved comparable accuracy to more intricate techniques such as [5]. MACER

uses this same method. Whereas MACER++[8], along with lines l-1, l and l+1, also uses those lines which contains the identifier reported in compiler error message.

For abstract code repair phase, TRACER considers the erroneous source line as sequence of tokens and attempts to predict new sequence of tokens as corrected target line using RNN.

In concretization step, abstracted symbols are converted back to concrete code using symbol table.

2.2 MACER

MACER[3] follows the same idea for error localisation as TRACER. To repair code, MACER uses 6 modular steps:

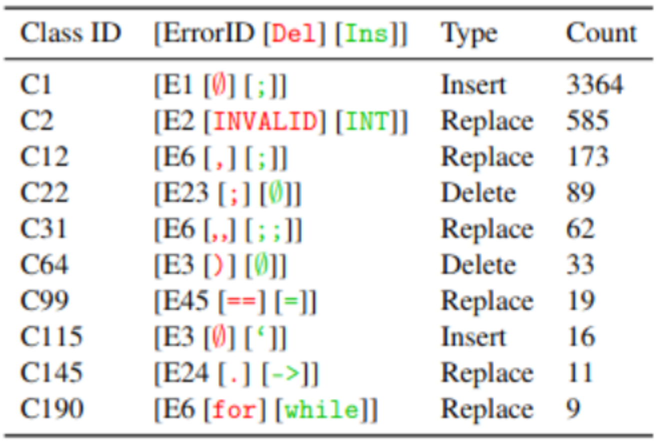
- repair lines localization
- feature encoding
- repair class prediction
- repair profile localization
- repair application
- repair concretization (undo abstraction)

As shown in [3], MACER has 1016 repair classes. It has 2239 dimensional feature vector(163 unigrams, 1960 bigrams and one more thing). MACER++ has around 700 repair classes. MACER considers compiler error ID for repair class creation whereas MACER++ does not. The benifit of seggregating the whole process into smaller modular steps is that it helps novice programmer to identify what modification can be done into their code to remove compilation error empowering them to learn how to address similar issues independently in the future. Many similar past works have been done like [5] [2] but they Provide repaired program directly, which may undermine the intended learning outcome. Such a code which merely compiles and doesn't resemble much with what the student wrote.

2.3 MACER++

Our method follows the same pipeline as of MACER++[8]. MACER++ proposed some optimizations to MACER like:-

- removing compiler error IDs information from repair classes
- better error localization by including those lines into potential candidate for erroroneous lines which contain those idenitfiers which were reported in compilation error. MACER only considered lines l-1, l and l+1 where l is the line reported as erroneous line by the compiler .



Class ID	[ErrorID [Del] [Ins]]	Type	Count
C1	[E1 [∅] [;]]	Insert	3364
C2	[E2 [INVALID] [INT]]	Replace	585
C12	[E6 [,] [;]]	Replace	173
C22	[E23 [;] [∅]]	Delete	89
C31	[E6 [,,] [;;]]	Replace	62
C64	[E3 [)] [∅]]	Delete	33
C99	[E45 [==] [=]]	Replace	19
C115	[E3 [∅] [']]	Insert	16
C145	[E24 [.] [->]]	Replace	11
C190	[E6 [for] [while]]	Replace	9

Figure 2.1: MACER's 1016 repair classes. Image credit [3]

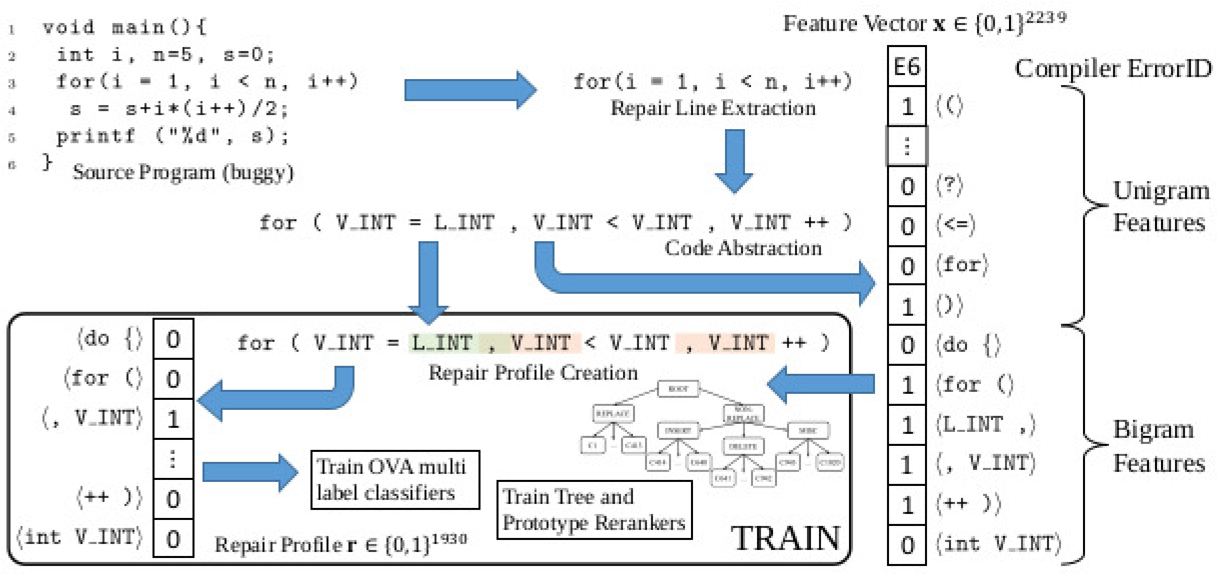
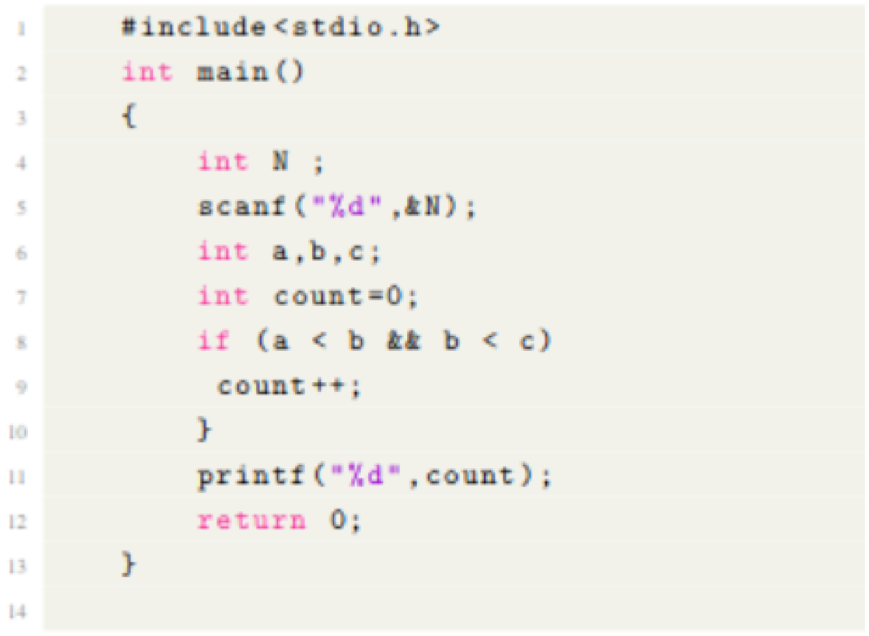


Figure 2.2: MACER pipeline. Image credit [3]

2.4 Deepfix

Deepfix[5] was one of the first approaches to use deep learning to automate program repair operations. By deliberately introducing faults into working programs, it creates a training dataset and then an attention[10] based sequence to sequence deep learning model is trained using this dataset.

As discussed in [5], during the repair process, DeepFix feeds the whole incorrect program rather than just specific lines to the deep learning model as input, in contrast to TRACER. The model then makes the necessary changes to the program, which is then compiled. If the repairs are successful, they are accepted. if not, the program is sent back into the model to fix any mistakes that could still exist. This is done iteratively until the program successfully compiles or hits a timeout limit.



```
1   #include<stdio.h>
2   int main()
3   {
4       int N ;
5       scanf("%d",&N);
6       int a,b,c;
7       int count=0;
8       if (a < b && b < c)
9        count++;
10      }
11      printf("%d",count);
12      return 0;
13  }
14
```

Figure 2.3: MACER++ error localization. Image credit [8]. Compiler gives the error:
13:1: error: extraneous closing brace('}')

2.5 Dr Repair

Dr.Repair[11] creates a program-feedback graph connecting diagnostic feedback and source code symbols relevant to program repair. Then, it further uses graph neural network for the purpose of applying repair.

As discussed in [11], it also uses a self-supervised learning strategy which generates a large number of additional program repair examples by using online unlabeled programs. Then it pre-trains its model using these examples.

In order to forecast error line indices, the model encrypts these inputs using LSTM and graph attention layers.

CHAPTER 3

Problem Formulation

Contents

3.1 Limitations of MACER and MACER++

We have built upon MACER++ by introducing two new operations viz. replace operation and move operation for the purpose of correcting erroneous program as well as to make error repair suggestion more easier to understand.

MACER and MACER++ support only insert and delete operations. These applications are able to detect replace operations only when number of symbols to be deleted is equal to number of symbols to be inserted. While this can be true in lot of cases in the dataset used by them but in general, it cannot always be true. Even when the number of symbols to be deleted is equal to number of symbols to be inserted, it may not be the case that replace operation needs to performed because those inserts and deletions may be required at different locations. Also, when number of inserts is not equal to number of delete, it may be the case that replace operations needs to be performed because more number of symbols may need to be replaced with less number of symbols and similarly less number of symbols might need to be replaced with more number of symbols. This helps our application to perform well in more general cases.

Similarly we have incorporated move operations in our application as well which MACER and MACER++ do not support. They handle move operations with delete followed by insert. This also helps the prediction model to be trained with compact data with same amount of information.

Suppose the symbol ';' needs to be moved from its current index in erroneous line to some other index to remove compilation error.

- The MACER++ repair class would look like: ['delete ;','insert ;']. This means delete the symbol ';' and insert the symbol ';' to remove compilation error from erroneous line.
- Our repair class would look like: ['move ;']. This means move the symbol ';' to remove compilation error from erroneous line.
- If there are many symbols which need to be moved then we can observe that MACER++ repair class would become lengthy and a bit difficult to understand by a student whereas our

repair class would contain the same information in a compact and easier to understand way. This could be used for better suggestions/hints that can be given to the students so that they can try to fix the error by themselves.

3.2 Restrictions with having only two operations

Only two operations viz. insert and delete were being used to handle vast number of repair cases by MACER and MACER++ which, in general, would not be able to repair some cases which can be repaired by introducing wider set of operations. Also, this offers more advantage for teaching and learning

CHAPTER 4

Proposed Method

Contents

MACER and MACER++ utilize Python's diff class library to identify insert and delete operations. They compare abstracted erroneous source lines with corresponding abstracted error-free lines from the training dataset to pinpoint symbols requiring insertion or deletion. However, we propose expanding beyond insert and delete operations by introducing replace and move operations. This broader set of operations can address additional repair scenarios and enhance our algorithm's performance across a wider range of cases.

4.1 Modified repair classes

A repair class tells the type of repair that should be applied on an erroneous line of code so as to remove compilation error. The benifit of it, as compared to other methods like Deepfix[5], is that it not doesn't only anyhow offers successful compilation but rather it suggests fixes that closely resemble with what student wrote.

As discussed in [8] [3], MACER incorporates the following data in its repair classes:

- Compiler Error ID
- list of tokens need to be inserted
- list of tokens need to be deleted

MACER++[8], as part of its optimization, argued that the use of compiler error IDs is not benificial and hence removed this information from repair classes and observed some improvements in prediction accuracy.
We have also not included the compiler generated error ID in our repair classes, just like MACER++.

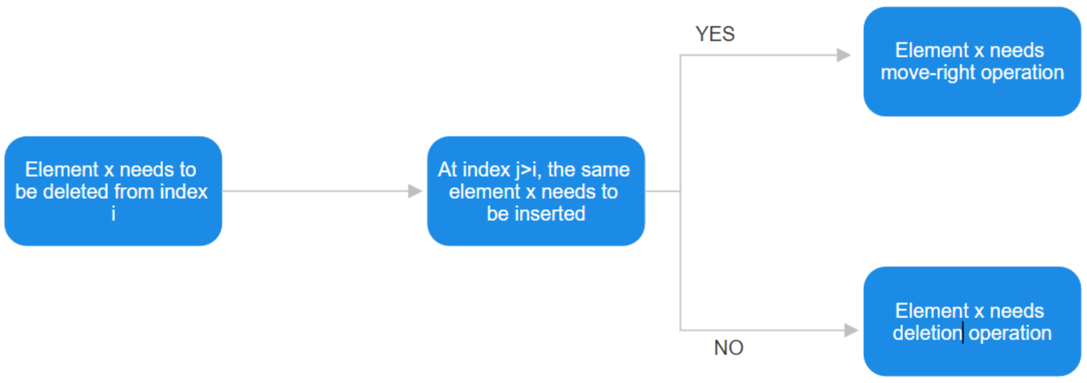

Element x needs to be deleted from index i
At index j>i, the same element x needs to be inserted
YES
Element x needs move-right operation
NO
Element x needs deletion operation

Figure 4.1: Detecting move right operation

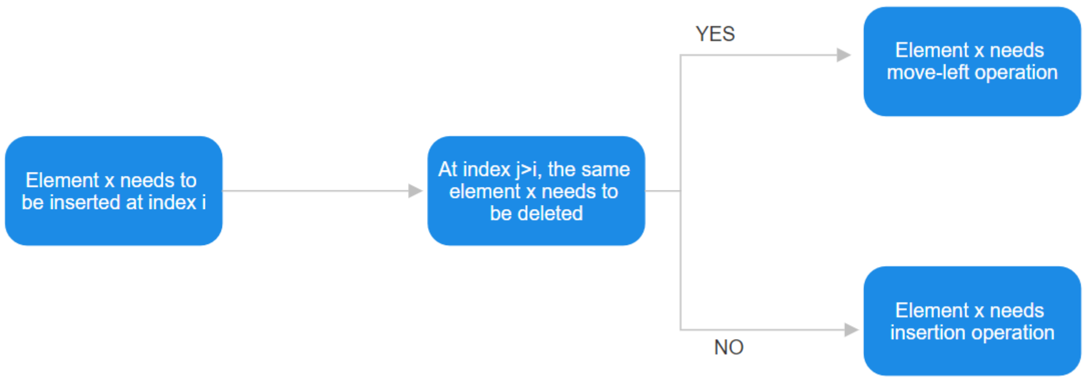

Element x needs to be inserted at index i
At index j>i, the same element x needs to be deleted
YES
Element x needs move-left operation
NO
Element x needs insertion operation

Figure 4.2: Detecting move left operation

Below are the types of repair classes in our proposed modification:

- **Insert repair class:** These are the repair classes in which all tokens should only be inserted. This is same as that of MACER++.
  Example: ['ins)\nins *\nins (']
  This tells that the tokens ')', '*' and '(' need to be inserted in the abstracted erroneous source line to remove compilation error.

- **Delete repair class:** These are the repair classes in which all tokens should only be deleted. This is same as that of MACER++.
  Example: ['del {', 'del }']
  This tells that the tokens '{' and '}' need to be deleted from the abstracted erroneous source line to remove compilation error.

- **Replace repair class:** These are the repair classes in which all tokens should only be replaced with some other tokens.
  Example: Consider the erroneous line:

if (fact (i) = < n & fact (i + 1) >= n)
The repair class for it will be:
['rpm =\nrpm <\nrpa <=']
This denotes that token '=<' needs to be replaced with '<=' from erroneous line of code to remove compilation error.

- **Move repair class:** These are the repair classes in which all the tokens should only be moved to some other location.
  Example: Consider the abstracted erroneous line:
  if (b <= 0) { b = 0 } ;
  The repair class for it will be:
  ['mve ;']
  This denotes that the token ';' needs to be moved either to its left or right from its current position in the erroneous line of code to remove compilation error.

- **Miscellaneous repair class:** These are the repair classes which require more than one type of repair (combination of insert, delete, replace and move repairs).

4.2 Repair application

- Repair classes are used to denote what type of tokens need to be inserted, deleted, replaces or moved and repair profiles are used to determine at which bigram, the edit needs to be done. These bigrams are nothing but bigrams of tokens present in the erroneous source line. Repair profiles are predicted using a One vs All classifier.

- We have done some modifications in repair application code of MACER++ to handle the new separate operations viz. replace and move that we have introduced in our work.

4.2.1 Move repair application

- In repair application of move class, we move the token which needs to moved (as predicted in our repair class) to all the positions to its left as well as to all the positions to its right and after each move, we check if the code is able to compile successfully. If it compiles, we stop. Otherwise, we move it to other position and check again.

- If there are multiple tokens that need to be moved, then we move the first token and for its each move step, we move all the other tokens recursively. At any point, when the code is able to compile successfully, we stop.

- The issue with this is that since this is a brute-force approach, the runtime complexity increases. Since we are performing repair application on abstracted erroneous line, the token which is predicted by the move repair class maybe present in multiple locations. So, we need to consider each of those tokens as our potential move candidate. This causes the run-time to increase even more.

- To avoid this, we observed in our dataset that students will not, in general, put an incorrect token to a large distance from where it is supposed to be. So, usually, a token would not need to be moved more than 3 indexes. So instead of moving each token to all the positions, we only move it 3 indexes to the left and 3 indexes to its right. We apply this on each token which needs to be moved.

4.2.2 Replace repair application

- In repair application of replace class, we take the token which needs to be replaced and we call it as toreplace. We call the token which should be put in the place of toreplace as withreplace.

- Since the repair application runs on abstracted erroneous line, there can be multiple instances of toreplace in the erroneous line. So, we need to consider each instance as the potential candidate for replace and replace each of them one by one and check which replacement causes the compilation error to go away.

- If there are multiple tokens which need to be replaced then rather than treating toreplace as a string, we create toreplace as a list of strings(similarly for withreplace). Then we replace each element of toreplace and then for each replacement, we replace the further elements of toreplace recursively. At any point, when compilation becomes successful, we stop.

4.2.3 Add repair application

- The add repair application is same as that of MACER++. All the tokens which need to be added are clubbed together and based on repair profile, they are inserted and checked if code compiles successfully. This was done after observing in the dataset that, in general, the tokens which student has missed to add, need to be added near each other.

4.2.4 Delete repair application

- The delete repair application is same as that of MACER++. All tokens to be deleted(as predicted by repair class) are one by one deleted from the positions predicted by the repair profile and checked if the code compiles successfully.

4.2.5 Miscellaneous repair application

- In miscellaneous repair application, sequence of delete, add, move and replace repair application is run and we check if compilation error goes away. We ensured that those operations which make use of repair profile like delete and add are run before those which do not make use of repair profile like replace and move.

- This is done because if we perform move or replace repair application first then the indices may get changed and we won't be able to make use of repair profile.

CHAPTER 5

Experiments

Contents

5.1 Dataset

We report our accuracy on TRACER's dataset[1]. In this dataset, there are compilation errors only on a single line, hence referred to as single line dataset as well. As discussed in [8] [3] [1], it contains 4326 problems. The dataset was gathered from first-year undergraduate students attending an introductory programming course at IIT Kanpur. Weekly programming tasks were required for this course, and its purpose was to assess students' programming skills. For every assignment, students were allowed to submit it more than once and only the final version was considered for grading. The students' attempts were saved periodically as they worked towards solving problems.

5.2 Detecting move-left/move-right vs move operation

5.2.1 move

advantage: more number of training examples will be available.
disadvantage: more time will be required in repair application because the symbol will be needed to move both left and right.

5.2.2 move-left and move-right

advantage: less time required in repair application(because move-left symbols will only be needed to move left and move-right symbols will only be needed to move right)

disadvantage: move training examples will be split into two parts(move left and move right). So, for each class(viz. move left and move right), the number of training examples will become less.

- We experimented with both move-left/move-right and move and we observed that it is better to use move operation rather than move-left/move-right operation.
- For example, if the repair class contains these information: move-left 'symbol1' , move-right 'symbol2' and move-right 'symbol1'. Note that symbol1 is present at two different locations and one of them needs to be moved left while other needs to be moved right. We won't be able to know which one of the symbol1 needs to be moved left and which one needs to be moved right.
- So, anyway we will need to check all the possibilities i.e. both the symbol1 should be moved left as well as right.
- This is exactly what we would end up doing if we had detected move rather than move-left/move-right and also, we are getting less number of training data for each move-left/move-right operation.
- Also, this case would be hitting more frequently as we are working on abstracted erroneous line where same type of symbols would be present multiple times.

5.3 Metric used

We have used **pred@k** metric, which was introduced by MACER[3], to evaluate our model's performance. This metric denotes the ratio of programs in test dataset on which the model was correctly able to repair and remove compilation error, to the total number of programs in the test dataset. Here k denotes the top k predictions of repair classes by our model.

5.4 Results

Here we report our result on pred@k metric where k denotes the top k repair class predictions.

pred@1	0.573
pred@2	0.641
pred@5	0.705
pred@7	0.708
pred@10	0.72

Table 5.1: pred@k measure

5.5 Less number of move training data

- we observed that move class is being predicted very rarely because of which some of the erroneous source lines which require moving a symbol to repair the compilation error are not getting repaired
- We observed that the number of training data points for move repair class is very less as compared to insert, delete and replace which may be cause of this issue.

5.5.1 Synthetic data generation for move class

- we wrote a script to generate synthetic move data
- from the TRACER's singleL training dataset, we took correct abstracted code and we moved one symbol to some other location in the same line that will cause compilation error.

5.5.2 Results after including synthetic move data

- we saw no change in result. Still, move classes were rarely being predicted.
- this could be caused due to the difference in nature of actual mistakes done by student and the type of mistake that we have introduced with respect to move repair.
- Instead of TRACER's single line dataset, we validated our move repair application on the synthetic move data generated by us that we kept aside for validation. We still observed that move classes were not being predicted on them by our model.

5.6 Performance on individual repair classes

Here we show pred@k measure on some top and lowest individual repair classes.

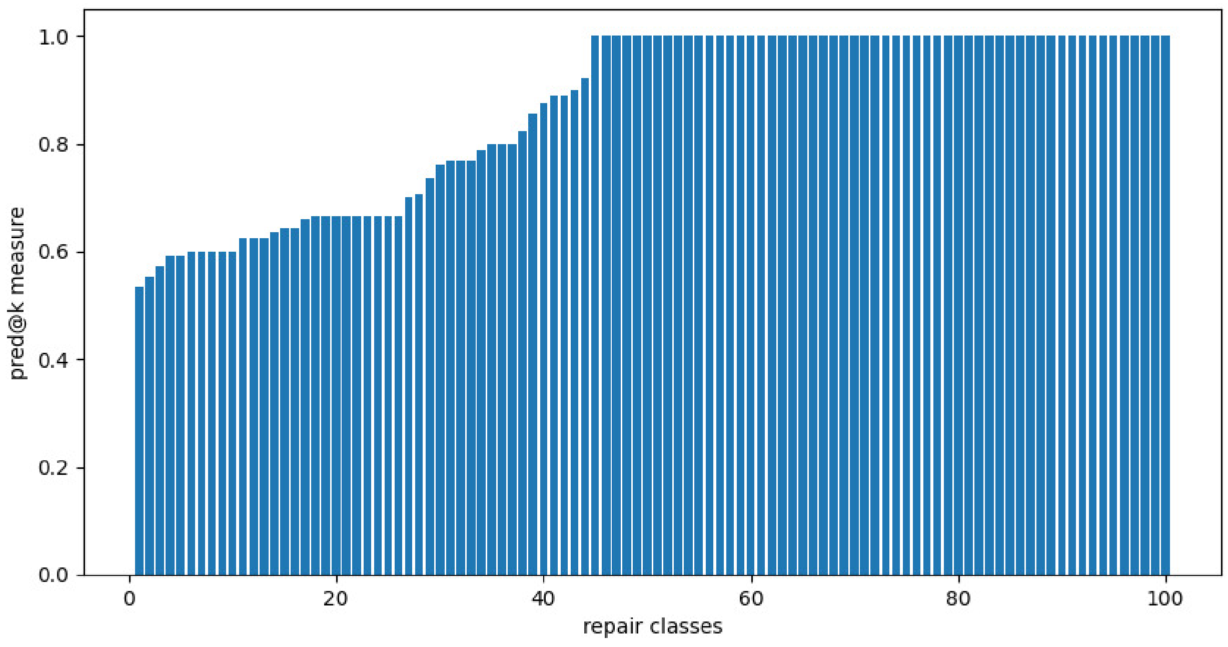


Figure 5.1: pred@k measure on top 100 repair classes

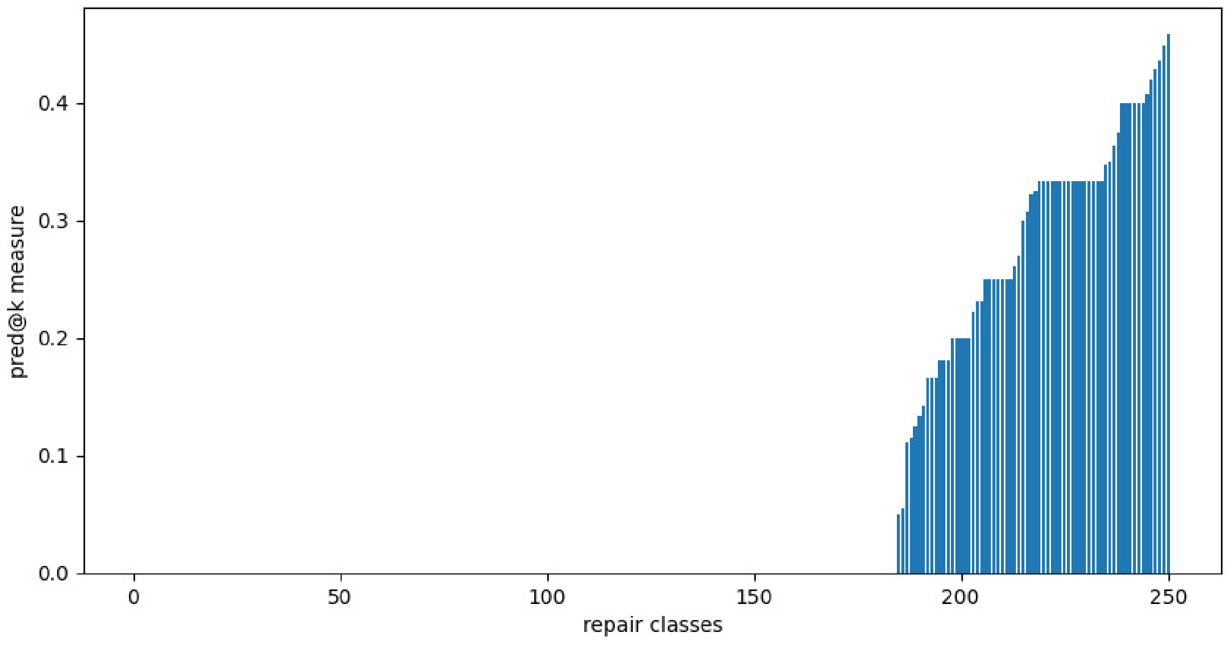


Figure 5.2: pred@k measure on lowest 250 repair classes

CHAPTER 6

PRIORITY: Event Logging

Contents

This chapter is an extension of the work [9]. Please refer to it for more details.

As shown in [8] [6], the primary objective of the PRIORITY[8][6] tool is to create a searchable index of a large number of programming problems collected from previous ESC101 course offerings. With the help of priority, in future offerings, the problem solvers of this course will be able to efficiently search for specific problems based on the programming skills, difficulty level, and concepts needed to solve them. By reviewing these past questions, problem solvers can gain insights into the pattern of questions present in the course, facilitating the process of creating new problems in a much easier and similar fashion.

6.1 Our contribution

We have added the event logging feature to priority. We will now take a look at how events are logged in PRIORITY using which we can collect data which can be used for further improving the ML algorithms that can be used to label new problems more accurately.

6.2 The setup

MySQL is an open-source relational database management system that uses SQL. It's one of the most popular database systems in the world and is commonly used for web applications, particularly those running on the LAMP (Linux, Apache, MySQL, PHP, Python, and Perl) stack. Integrating a MySQL data model into a Node.js application involves the following steps:

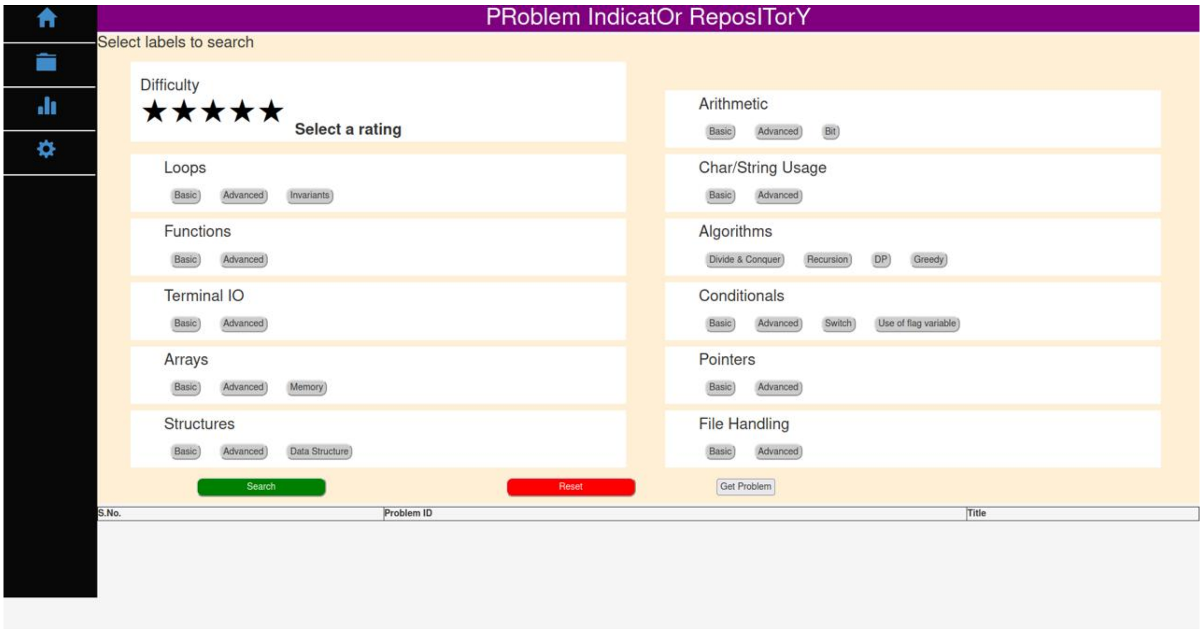


Figure 6.1: PRIORITY. Image credits: [9]

1. **Install Dependencies** First, we need to install the necessary packages. We need the MySQL package to connect to the MySQL database and interact with it.

2. **Set Up Connection** We need to establish a connection to the MySQL database from the Node.js application. This involves providing connection details such as host, user, password, and database name.

3. **Perform Database Operations** After establishing a connection, we can perform various database operations like query, insert, update, and delete.

4. **Close Connection** After we are done with our database operations, it's a good practice to close the connection to release resources.

6.3 First Logging Event

PRIORITY logs events in a MySQL database. The first logging events occurs when a user performs a search in PRIORITY.

6.3.1 Process

Let us explain this event in detail in a sequential manner.

1. user opens PRIORITY and sees the homepage.

2. user selects a query. In this case, a query means that a user selects a number of labels (one or more) and difficulty level (Optional).

3. User clicks on Search button.

4. The user sees a list of questions on the page.

After step 4, the first event is logged.

6.3.2 Log Data

The log event contains the following information.

- **TimeStamp:** The current data and time of the system when the Search button was clicked. Datatype is datetime in NodeJS.
- **Username:** The PRUTOR[4] login id of the user who has performed the search request. Datatype is string in NodeJS.
- **Problem Labels & Difficulty:** The selected labels are put in a list. At the end of the list, if the user has selected a difficulty level, a single character 1,2,3,4 or 5 is appended denoting the difficulty level. In NodeJS, it is an array of strings.
- **Search Results:** The list of questions received in reponse to the query are saved in the form of problem ids of each question. This is stored as an array of problem ids in NodeJS.

6.3.3 Uses & Benefits

This type of log will serve 2 purposes mainly.

- Creating a repository of frequently searched topics on PRIORITY. This log can further be classified based on username, user-role (Tutor / Instructor) or timestamp. It can be used to derive crucial inferences about the user preferences on using PRIORITY.
- The list of responses received in a particular query can help in improving the indexing and searching process when the database grows bigger with passage of time.

6.4 Second Logging Event

The second logging events occurs when a user imports the problem from PRIORITY to PRUTOR. Logging this event serves the benifit of providing user preference when a list of questions is provided. These logs can be used for further analysis and improvement.

6.4.1 Process

Let us explain this event in detail in a sequential manner.

1. User views a problem on PRIORITY.
2. User finds that the questions is suitable for current offering of ESC101.
3. User clicks on Import button.

6.4.2 Log Data

The log event contains the following information.

- TimeStamp: The current data and time of the system when the Search button was clicked. Datatype is datetime in NodeJS.
- Username: The PRUTOR login id of the user who has performed the search request. Datatype is string in NodeJS.
- PRUTOR Problem ID: This field corresponds to the new problem id for which user wants to import a problem from PRIORITY. It is the problem id of the new question in PRUTOR database.
- PRIORITY Problem ID: This field corresponds to the problem id field of the problem which is being imported to PRUTOR.

6.4.3 Uses & Benefits

This type of log will serve many purposes. Some of them are listed below.

- It will provide a direct relationship between questions created on PRUTOR as part of ESC101 course.
- It will provide a check to question duplication. If a tutor directly copies the problem from PRIORITY without changing anything. The tutor can observe this easily.

Bibliography

[1] Umair Z. Ahmed, Pawan Kumar, Amey Karkare, Purushottam Kar, and Sumit Gulwani. Compilation error repair: for the student programs, from the student programs. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*, ICSE-SEET '18, page 78–87, New York, NY, USA, 2018. Association for Computing Machinery.

[2] Sahil Bhatia and Rishabh Singh. Automated correction for syntax errors in programming assignments using recurrent neural networks, 2016.

[3] Darshak Chhatbar, Umair Ahmed, and Purushottam Kar. *MACER: A Modular Framework for Accelerated Compilation Error Repair*, pages 106–117. 06 2020.

[4] Rajdeep Das, Umair Z. Ahmed, Amey Karkare, and Sumit Gulwani. Prutor: A system for tutoring cs1 and collecting student programs for analysis, 2016.

[5] Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish K. Shevade. Deepfix: Fixing common c language errors by deep learning. In *AAAI Conference on Artificial Intelligence*, 2017.

[6] Sharath Hp. Real world deployments of ai-assisted tools for compilation error repair and program retrieval, 2021.

[7] Jeet Sarangi. Program repair and retrieval on the prutor platform - ii, 2024.

[8] Fahad Shaikh. Advancements in ai-assisted compilation error repair and program retrieval, 2021.

[9] Utkarsh Srivastava. Program repair and retrieval on the prutor platform - iii, 2024.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[11] Michihiro Yasunaga and Percy Liang. Graph-based, self-supervised program repair from diagnostic feedback, 2020.