Overview

MOOC Courses for introductory topics
- Vast solution set
- Vast incorrect set
- Too many students to individually review submissions

SARFGEN

- Within 2 seconds: Provide useful automated feedback for 89.7% of incorrect solution
- Apply minimal repair to an incorrect solution using existing solutions
- More reliable and amenable than existing frameworks
- Fully automated
- Minimal Repair
- Unrestricted repair
- Portable

Comparison vs existing solutions

<table>
<thead>
<tr>
<th>Approach</th>
<th>No Manual Effort</th>
<th>Minimal Repair</th>
<th>Complex Repairs</th>
<th>Data-Driven</th>
<th>Production Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoGrader</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CLARA [10]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>GLASS [11]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>KLG [21]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>RIFAZER [2]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CoderAsst  [5]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>SARFGEN</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Concerns

- Lead Students in the nearest correct direction
- Provide near-instantaneous feedback
Example: Print Chessboard

Teach students conditional and loops constructs

Example Incorrect Solutions

Example Repair

Challenges

Chessboard

Algorithm Overview

- SARFGEN generated feedback within 2 seconds
- Selected solution most similar to submission
- Feedback consisting of:
  - Number of changes
  - Location of error (Line number)
  - erroneous expression that needs to be altered
  - value of new (sub-)expression
- Feedback is customizable amongst the 5 different kinds of information

- Problems can be solved using many different algorithms
- 337 Different Control Flow structures for the chessboard problem
- Cannot simply modify program to fit solution
- Must modify using as few repairs as possible

Algorithm Overview

- Implemented in C# and uses Microsoft’s Roslyn Compiler
- Dell XPS used for the experiment

- Searcher - (INPUT) = an incorrect program
  - Finds top k closest solutions among correct programs
- Aligner - (INPUT) = previously found closest solution
  - Align candidate programs to incorrect program and produce discrepancy list
- Repairer - (INPUT) = previously aligned set
  - Optimized by removing the syntactic/semantic redundancies
Search

- Perform exact matching with the incorrect program and solution set.
- Control Flow:
  - Search CF set using a syntactic matching method instead of dynamic approach.
  - Embed AST into position aware characteristic vectors.
  - Runtime of the search portion is: $O(m + \rho \ast (|h_{b_1}|, ..., |h_{b_L}|)^{2q-1})$.

Align

- Dynamic approach of comparing the runtime issues for each variables suffers from the same scalability issues.
- Align using position aware characteristic vectors.
- Two-level statement matching:
  - Align using position aware characteristic vectors.
  - Pick matching that minimizes distance.

Repair

- 3 types of fixes needed:
  - Insertion Fix
  - Deletion Fix
  - Modification Fix
- Computed set of differences can be placed in two categories:
  - Syntactic differences
  - Semantic differences
- Repair filters benign differences and is optimized so the set of differences does not explode in size.

Critique

- Dependent on the data that was used.
- No general API so had to port over all code into C to compare to CLARA.
- Has difficulty dealing with corner case programs.
- Every problem has a wide set of solutions, solution set may not have them all.
- Still has issues with scaling, although they try to play it off.

Results

- Problem sets included chess board printing problems and more.
- Generates feedback for 4311 out of 4806 programs within 2 seconds.
- Comparison vs CLARA:
  - CLARA better for small programs, does not scale well.

Related Work: Automated Feedback Generation

AutoGrader:
- Uses a manually created error model for corrections.
- Cannot perform complex repairs: adding, deleting, swapping statements.

Clara:
- Employs a correct solution set to generate trace-based repair.
- Doesn’t account for new correct execution traces.
- Adds unnecessary repairs for new correct solutions.
Related Work: Automated Feedback Generation

**skp:**
- uses deep learning to model code fragments
- each program is a collection of code fragments
- must be retrained for new problems
- can only apply syntactic analysis

**QLOSE:**
- measures program distance syntactically and semantically
- uses predefined template for repairs

**REFAZER:**
- learns a syntactic transformation pattern from examples of statement/expression instances
- suffers from same issue as QLOSE
- large amount of incorrect programs

**CoderAssist:**
- clusters assignments based on solution strategy
- requires manual instructor intervention
- however does provide feedback

Related Work: Automated Program Repair

**Gopinath et al.**
- SAT based approach for repairing buggy programs
- encode buggy constraint to SAT constraint

**Konighofer and Bloem:**
- localize faulty components with model-based diagnosis
- produce results based on SMT reasoning
- only considers RHS of the assignment statements

**Prophet:**
- probabilistic, application-independent model
- generates a set of successful human patches

**Program mutation, genetic Programming:**
- repeatedly mutate statements ranked by suspiciousness
- highly inefficient: large space of mutations $10^{12}$

Related Work: Automated Debugging and Fault Localization

**Delta Debugging / QuickXplain:**
- ranks likely fixed prior to dynamic analysis
- exploits the initialization loop

**Jose and Majumdar:**
- error localization using Max-SAT
- suffers from limited capability in producing false

Questions?