Automated Debugging

Arnav Gupta

April 21, 2024

Contents

1 Statistical Fault Localization

Assign scores to program statements based on their occurrence in passing or failing tests. Assume correlation equals causation.

$$\text{score} = \frac{\frac{\text{fail}(s)}{\text{all fails}}}{\frac{\text{fail}(s)}{\text{all fails}} + \frac{\text{pass}(s)}{\text{all passes}}}$$

where $\text{fail}(s)$ is the number of failing executions in which s occurs and $\text{pass}(s)$ is the number of passing executions in which s occurs.

High suspiciousness score means executed mostly by failing test.

2 Delta Debugging

With test all, every commit is tested individually. With batch testing, commits are tested in groups. If tests pass, merge the commits.

Once a program failure is reproduced, figure out what is relevant and simplify.

This is important for ease of communication, easier debugging, and identifying duplicates.

To find problems, use binary search (cut test case in half) and iterate. Throw away half the input and see if output is still wrong. If not, go back to the previous state and discard the other half of the input. Final region that cannot be iterated upon (all correct) is simplified input (conflicting solutions).

With finer input graularity, chance of finding a failing input subset is higher and progress of search is slower. Start with few and large changes and more to more, smaller changes.

Let R be the set of possible inputs, where $r_p \in R$ corresponds to an input that passes, and $r_f \in R$ corresponds to an input that fails. Can go from one input $r1$ to another input $r2$ by a series of changes, where a change δ is a mapping $R \to R$ that takes 1 input and changes it to another input.

A change δ can be decomposed to a number of elementary changes $\delta_1, \delta_2, \ldots, \delta_n$ where $\delta = \delta_1 \circ \delta_2 \circ \cdots \circ \delta_n$ and $(\delta_1 \circ \delta_j)(r) = \delta_j(\delta_i(r))$. By composing the deletion of single characters, can get a change that deletes part of the input file.

An input wth failure is an input without failure with changes c_f made on it. Subsets of these changes are test cases.

Given a test case c, want to know if the input generated by applying changes in c to r_p causes the same failure as r_f. Define the function `test` that takes an element from the powerset of c_f and gives either P, F, or ?, such that it returns F iff applying those changes to r_p gives a failing input.

To minimize test cases, find the smallest test case c such that `test(c)` is F. A failing test case $c \subseteq c_f$ is called the global minimum of c_f if for all subsets c' of c_f, if c' is smaller than c, then `test(c')` is not F. Local minimum if c' is a proper subset of c.

The **global minimum** is the smallest set of changes which will make a program fail. Finding the global minimum may require performing an exponential number of tests.

1-minimality is to find a set of changes that cause the failure, but remove any change that cases the failure to go away.

A failing test case $c \subseteq c_f$ is n-minimal if for all $c' \subset c$, if $|c| - |c'| \leq n$, then `test(c')` is not F.

A failing test case is 1-minimal if removing one change, makes the test not fail anymore.

To find a 1-minimal subset of c: check if c is 1 minimal, if not, recurse on $c - \{\delta\}$ for some $\delta \in c$ where running `test` on $c - \{\delta\}$ gives F.

Worst case runtime is $O(N^2)$.

Can do better than this by dividing the change set in 2 initially, increase the number of subsets if can't make progress, and if lucky, search will converge quickly.

Delta debugging algorithm finds a 1-minimal test case.

1. Partition the set c_f to $\Delta_1, \Delta_2, \dots, \Delta_n$, where $\Delta_1, \Delta_2, \dots, \Delta_n$ are pairwise disjoint, and $c_f = \Delta_1 \cup \Delta_2 \cup \cdots \cup \Delta_n$.
2. Define the complement of Δ_i as $\nabla = c_f - \Delta_i$.
3. Start with $n = 2$.
4. Test each test case defined by each partition and its complement.
5. Reduce the test case if a smaller failure inducing set is found, otherwise it refines the partition, so $n = n \cdot 2$.

Worst case is still quadratic, if subdividing until each set is of size 1 (reduced to naive algorithm). But for single failure, converges in $\log(N)$, so binary search again.

Delta debugging is a technique, not tool. Probably must be re-implemented for each significant system to exploit knowledge changes. However, relatively simple algorithm with big payoff, worth re-implementing.

Can apply delta debugging with batch testing and bisect. Git bisect performs a binary search to find the faulty commit.

In practive testing takes too long and is too expensive, so only test every X commit. If pass, merge all. If fail, bisect to find the issue. In the worst case, git bisect looks at $\ln(n)$ commits.