Non-conventional Testing Techniques

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

02/03/2025

Kira Chan

https://cse.msu.edu/~chanken1/

How do we test our code?

- Written test?
  - You have some domain knowledge, and you are using them to test your program
  - You can independent verification
  - Random people on the street?
- Relatively easy to test if you have 1 input parameter (is it?)
- 2 parameters? R^2
- 3 parameters? R^3
- 4 parameters? R^4
- …

Test cases

- A successful test case is one that fails (catches an error)
  - Why?
- If you do not know that if a successful test case means
  - 1) Your test case is bad
  - 2) Your code is bad

A software tester walks into a bar

- Runs into a bar.
- Crawls into a bar.
- Dances into a bar.
- Flies into a bar.
- Jumps into a bar.
- And orders:
- a beer.
- 2 beers.
- 0 beers.
- 99999999 beers.
- a lizard in a beer glass.
- -1 beer.
- "qwertyuiop" beers.
- Testing complete.

- A real customer walks into the bar and asks where the bathroom is.
- The bar goes up in flames.

Testing is hard

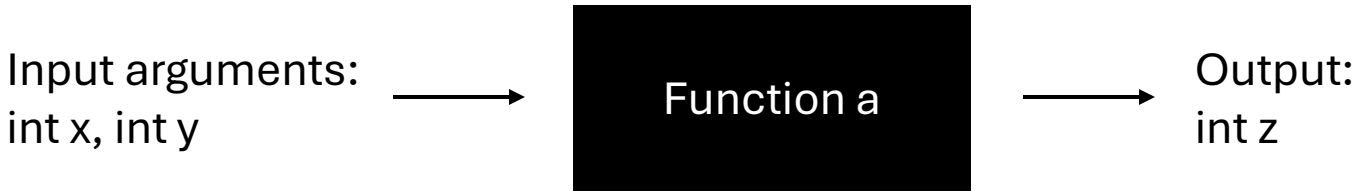
- Humans test with bias
- When you test your code, you test it *gently*

Towards Automated Testing

- Can we automate some testing?
- Some things we will cover today
- Fuzz testing (fuzzing)
- Evolutionary Search-based testing

Black-box model of a function

- Remember the concept of black-box testing?
- Given some input, transform it into outputs.



Some magic occurs here

Fuzzing

- Remember the concept of black-box testing?
- Given some input, transform it into outputs.
- What if we provide invalid, unexpected, or random data as input to monitor for crashed
- Developed by University of Wisconsin – Madison Professor Barton Miller
  - Uses external “noise” to test if a system is tolerant
- Found that traditional UNIX, mac, and Windows programs would routinely crash with unexpected inputs

Types of fuzzing

- Coverage-guided fuzzing
  - Tracks “code coverage”
  - Trigger all the code logic to make sure output is right
- Mutation-based fuzzing
  - Modifies existing data and input to find crashes
  - Advantages: you know what input crashed it, you can trace it

Use cases

- Incorrect behaviours
- Input errors and crashes
- Can catch memory issues
  - Memory leaks
  - Buffer overflows
  - Use after frees
  - Stack overflows
  - Crashes, etc.
- Security issues
- Very good at finding odd programming errors

Evolutionary Inspired Search Testing

- Leverage things we see in nature
- Particle Swarm Optimisation
- Ant Colony Optimisation

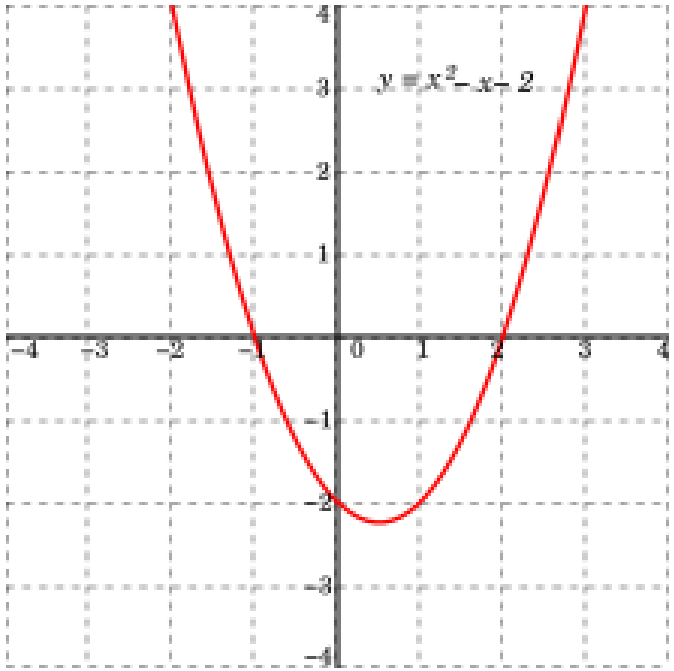
Skipped a lot of details

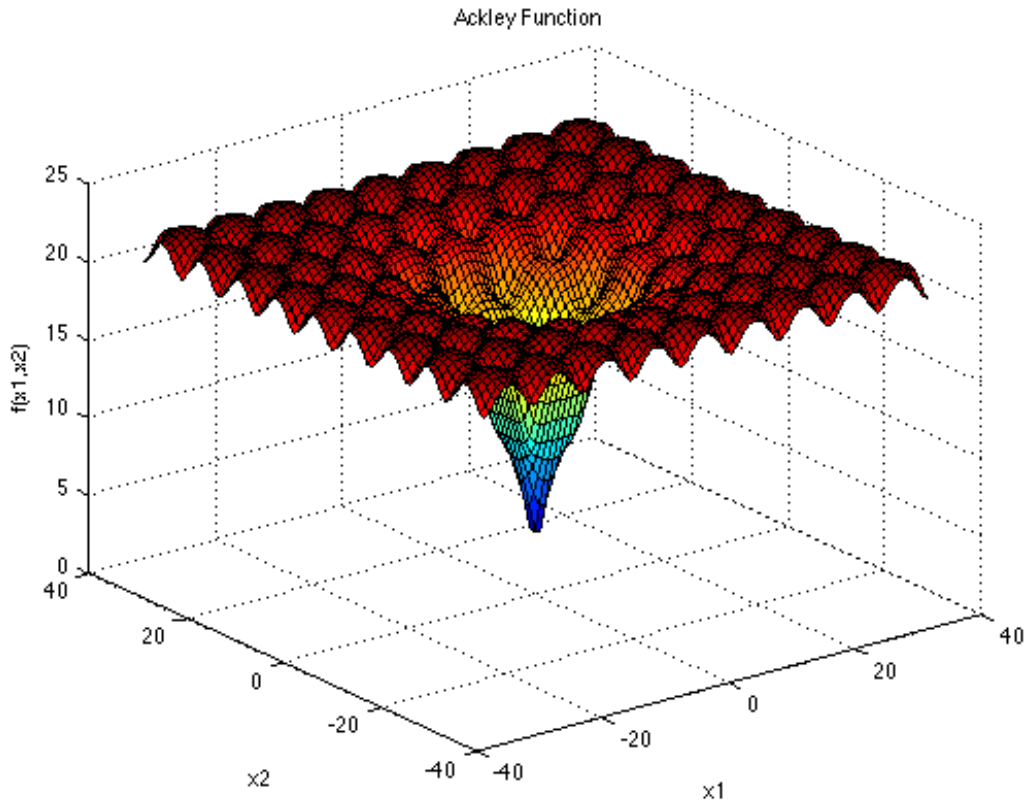
- I am leaving out a lot of details here, otherwise we will spend a semester on this
- If you are interested, then this is definitely grad school material
- We have a course **CSE848 Evolutionary Computation** taught by Dr. Wolfgang Banzhaf
  - Very big figure in the evolutionary computing theory
- MSU actually as a lot of famous people in this field

Evolutionary Search-based Testing

- Inspired by Darwinian Evolution
- Map solutions to a ``genome''
- Use nature inspired techniques to evolve solutions (a population)
- Mutation, crossover, and selection mechanism
- Evaluate an individual based on how well they perform (i.e., how close to error)
- Individuals who have high scores get to reproduce
- Slowly get to the solution

Solving for optimum





On searching

- I am going to ``punt'' the search stuff later
- I will cover how most problem are really just a search / optimisation problem in principle

Completeness

- When do you stop testing?

- These techniques are not meant to replace traditional testing
- They are complementary, often meant to help find bugs you otherwise would not due to developer bias

- You can make the code during the day, and fuzz test it overnight

Person of the Day
John Henry Holland

- Pioneer of Evolutionary Computing field
- Introduced genetic algorithms in 1960s
- Received his M.S. and Ph.D. at University of Michigan and have been a professor there