Sikraken Optimization Design Manual

Project by: Kacper Krakowiak (C00271692)
Project Supervisor: Dr Chris Meudec

Table of Contents:

Technologies Used:

1) Linux OS
2) C
3) Python
4) Genetic Algorithm
5) Sikraken
6) Eclipse (Language)
7) PTC-Solver
8) Testcomp
9) TestCov
10) BenchExec
11) GitHub

Sikraken is designed to run on a linux based system, so all development and work will be made on that mentioned Operating system. Sikraken is designed to generate test cases for C code, so the User Assist Tool will be written in C just to keep things consistent.

Introduction:

The Sikraken Optimization Tool is designed to enhance the efficiency of test case generation for C code samples, by implementing a Genetic Algorithm approach to parameter optimization. Sikraken, is a powerful tool for generating test cases for C programs, but in its original state, requires manual parameter tuning to achieve optimal results. This project aims to create an intelligent algorithm that will automate the parameter selection process, improving code coverage while minimizing execution time.

The key objectives of this tool are:

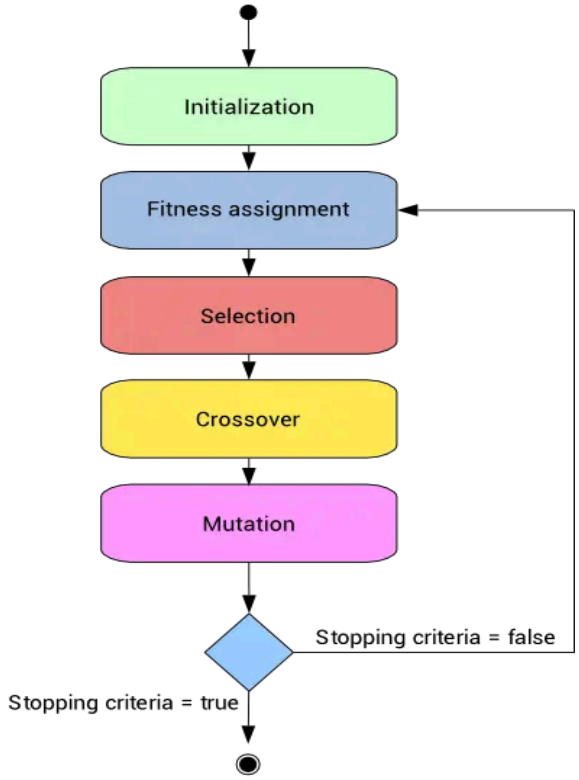
- Automate the parameter selection process
- Maximize code coverage for test cases
- Minimize execution time
- Output results on a graph
- Support parallel processing for faster optimization

Problem in Regression mode:

Sikraken has two modes, budget and regression. In budget mode we set a time value at which Sikrkaen will cease to generate test cases, when reached. In regression the user has to manually input 2 values (restarts, tries), which correspond to how Sikraken will behave when it gets stuck in a loop or a conditional statement in a given C code sample. Currently Sikraken doesn't have a way of finding the best combination of these two integers and the user has to guess and run the program multiple times and determine the best combination through looking at CPU run time and number of tests generated from the separate log file.

Genetic Algorithm:

The genetic algorithm is present in nature, following Charles Darwin's theory of natural selection. The definition can be summarised as “principle by which each slight variation [of a trait], if useful, is preserved”. Basically means the strongest trait passed down from a set of parents will continue to multiply, and the weaker traits will inevitably die and wont make it to the multiplication phase. There are a few nuances, such as mutation and crossover. Mutation is self explanatory, in further (or more refined) generations, some children will undergo random changes outside of what has so far evolved in a given pool (very small chance of mutation). Just like in nature all living organisms features are written in DNA through Genotypes and Phenotypes, we can actually simulate these in computing.



The Knapsack Problem:

Lets say that we have a bag with limited weight capacity, and we have multiple items. Each item has 2 attributes, weight and value. We want to find the best combination of items to maximise the weight used (without going over the allowed limit) and use items with the highest value. This problem can be solved easily through randomness if there is a limited number of “items”, but once the total number of items increases the total combinations and possibilities rises exponentially. This is actually very similar to what Sikraken aims to figure out in regression mode. 2 attributes, find the best combination of two attributes (restarts, tries) within a limited constraint of two values, tests generated(higher the better), and CPU run time (lower the better).

Core Components

1. Python

- Implements the genetic algorithm
- Manages population evolution
- Tracks fitness metrics
- Handles parallel execution (built in to Python language)

2. Integration Layer (Actual Algorithm)

- Interfaces with Sikraken test generator
- Manages parameter passing
- Processes Sikraken outputs
- Integrates with TestCov for coverage analysis (regex)

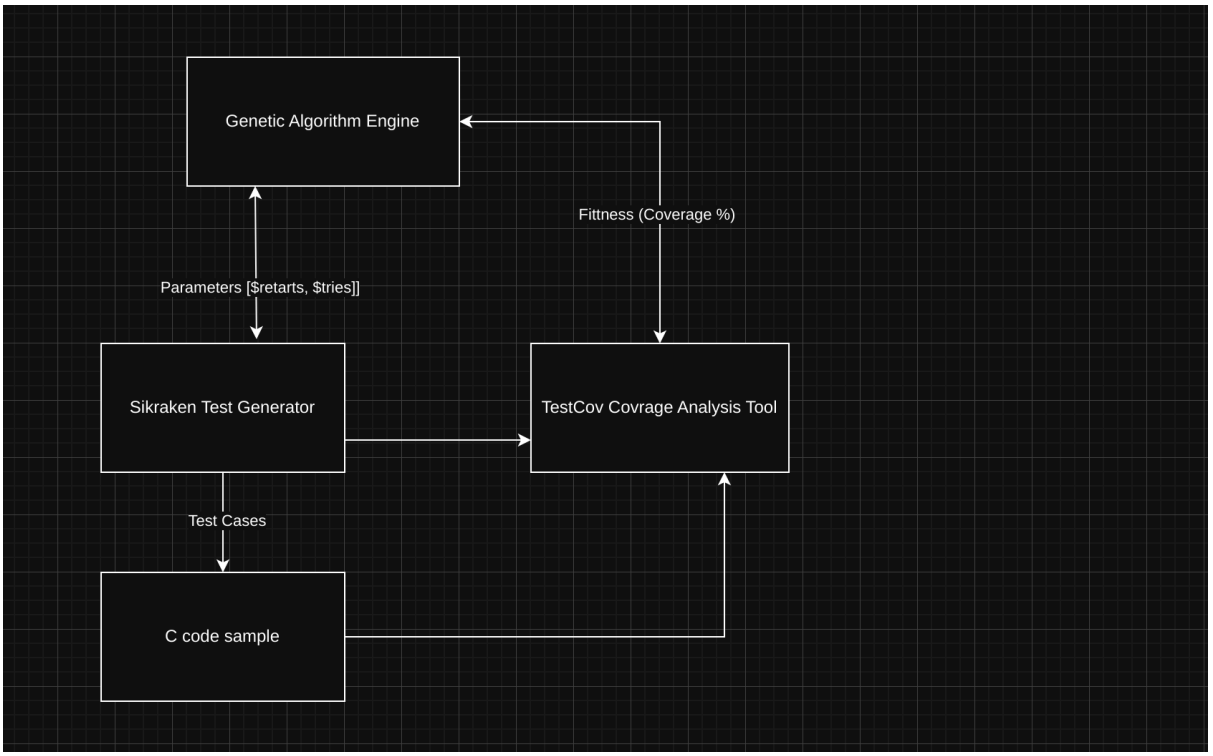
3. Interface

- Command-line interface
- Results visualization
- Parameter recommendations
- Progress monitoring

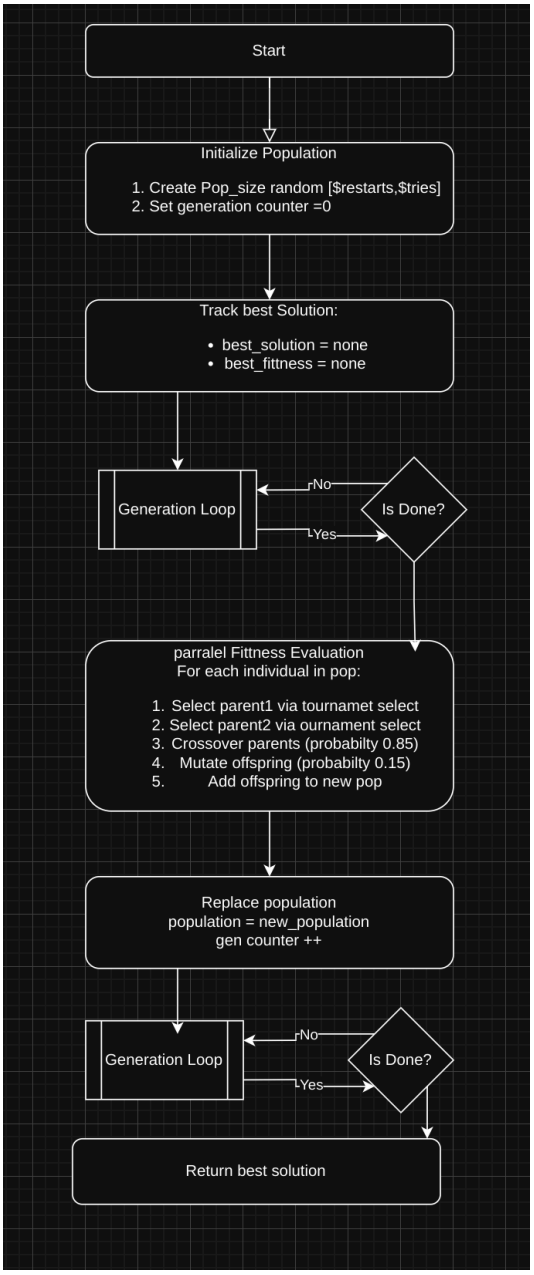
Data Flow:

- User inputs C code file to test
- Optimization engine generates initial parameter population
- Parameters are passed to Sikraken for test generation
- TestCov analyzes coverage of generated tests
- Results feed back to the optimization engine
- Process iterates until optimal parameters are found
- presented to user

High Level Diagram:



Low Level Flow Diagram:



Genetic Algorithm in Sikraken User Assist Tool:

1. Initialization
   The algorithm will start by creating an initial random population of individuals. The “individuals” will refer to $restarts,$tries integer pair. The very first pair will be completely random.
2. Evaluation
   Next an evaluation will occur, where the results for the initial population are read from the log file (test_run_Problem03_label00.log) through regex (only pulling the relevant CPU time and Tests Generated number, which will become the “fitness” or a means of our algorithm knowing what to achieve)
3. Selection
   Choosing of individuals from the population with the best results (fitness)
4. Cross-over
   The algorithm randomly assignins a cross-over point and mixing two parents to form a new offspring. For example we may generate parent1:[20,30] and parent2:[15,18]. The algorithm will choose a random swap so we could have offspring 1:[18,20], or [30,15] or other combinations. This will allow us to create diversity and explore more solutions, and by mixing parents (which were chosen in the previous operation to be the strongest pair, just like in nature)
5. Mutation
   Occurs (by chance), where a random integer may be introduced to the population evaluation, but more than likely it will flop
6. Evaluation (next iteration)
   The offspring will also measure its fitness and compare that of the parent. The set of offsprings with a higher fitness score will be favoured to replace the current parent population
7. Parallelization
   Multiple parameter evaluations run concurrently, where the Thread pool manages parallel execution. A locking mechanism prevents resource conflicts with TestCov, that does not work in parallel
8. Finalization
   After a set of generations (or iterations of population) the highest remaining $restarts$tries pair with the highest fitness score will be selected as the optimal regression input pair
9. Visualization
   A graph of the entire algorithms progress during execution is outputted to the user, for more informed and calculated display

Parallelization Strategy:

The tool implements a hybrid parallelization approach:

- Sikraken processes run in parallel across multiple threads
- TestCov runs are serialized with locks to prevent conflicts
- Results processing occurs concurrently

This strategy provides a significant speedup compared to sequential execution, dramatically reducing the time required to find optimal parameters.

Presource Management:

- Dynamic thread allocation based on available CPU cores
- Memory-efficient data structures for tracking populations (standard in Python)
- Timeout mechanisms to handle hanging processes
- Error handling to recover from failed evaluations

Error Handling:

- Parameters are halved (for both $restarts,$tries integers), for more efficient timeout recovery as opposed to randomizing anew
- A timeout value set, to prevent overly large integer pair from wasting processing time
- Printing to the terminal, to let the user know of a failed integer pair

References:

Natural-Selection_Wikipedia(2024). Research strategy [online]. Available from: https://en.wikipedia.org/wiki/Natural_selection#:~:text=He%20defined%20natural%20selection%20as,likely%20to%20survive%20and%20reproduce.[accessed 6 December 2024].

What-Are-Genetic-Algorithms_YoutTube(2024). Research strategy [Video]. Available from:https://www.youtube.com/watch?v=XP2sFzp2Rig[accessed 6 December 2024].

Genetic-Algorithms-Explaine-By-Example_YoutTube(2024). Research strategy [Video]. Available https://www.youtube.com/watch?v=uQj5UNhCPuo[accessed 6 December 2024].

Knapsack-Problem_Wikipedia(2024). Research strategy [online]. Available https://en.wikipedia.org/wiki/Knapsack_problem[accessed 6 December 2024].