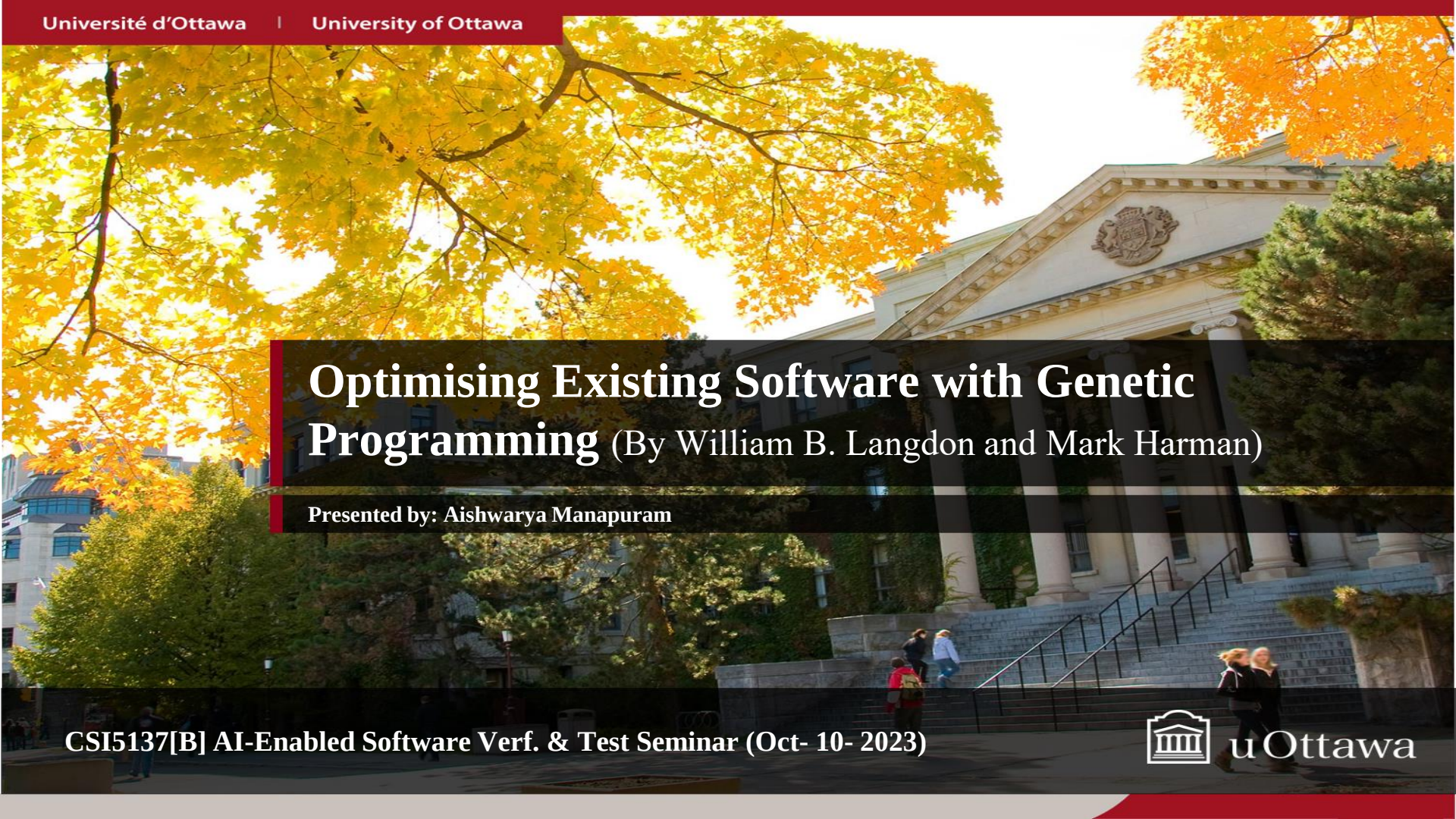


# Optimising Existing Software with Genetic Programming (By William B. Langdon and Mark Harman)

**Presented by: Aishwarya Manapuram**

# Agenda

- Introduction
- Problem Statement
- Solution (GISMOE)
- MOTIVATION
- Implementation
- Evaluations
- Results
- Critique
- Conclusion
- References

# Introduction

- Optimizing existing software means **making improvements** to the software to **enhance** its **performance, efficiency, and/or functionality**. Optimization aims to make the software run faster, use fewer resources (such as memory or CPU), and deliver a better user experience.

- The paper introduces the concept of gen**etic improvement**, which is the **process of automatically improving a software system's behavior using genetic programming**.

# Problem Statement

- The authors of this paper aim to show that genetic programming techniques can be applied to real-world, highly complex existing software systems to **enhance non-functional properties** of a system, such as **execution time and resource consumption**, while maintaining or improving its functional properties.

- **Real-World, Complex Software:** Bowtie2, a widely-used DNA sequencing system with approximately 50,000 lines of C++ code, as their target system.

- **Functional Correctness**: While optimizing non-functional properties, the authors also aim to ensure that the evolved software maintains or enhances its functional correctness. **The new code should produce results that are at least as good, if not better, than the original code**.

# Solution

- The **goal** of GI is to automate as much of the improvement process as possible. Thus, new implementations can be discovered by an **evolutionary process**, rather than being hand-crafted by human programmers.

- **Automated Test Oracle:** The genetic improvement process relies on a set of test cases obtained from running the original system. An automated test oracle is used to compare the output of the original and evolved code.

# Solution - GISMOE

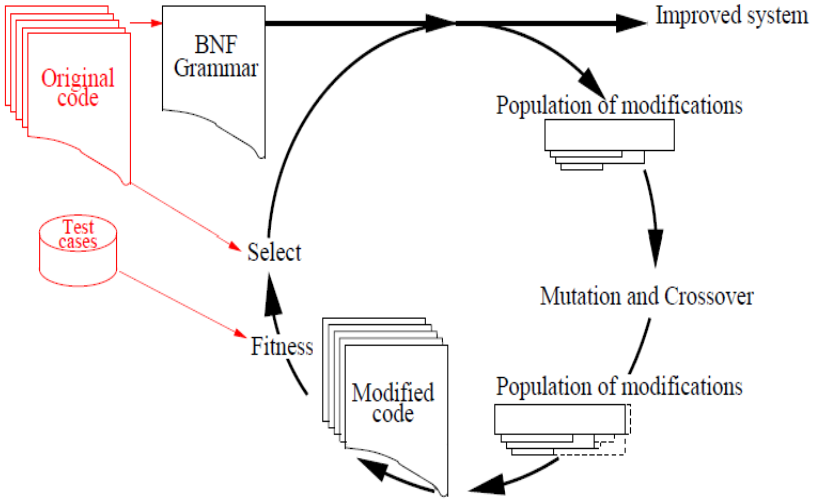
**Major components of GISMOE approach**

Left: system to be improved and its test suite.

Right: genetic programming optimises modifications which originate from a grammar that describes the original system line by line.

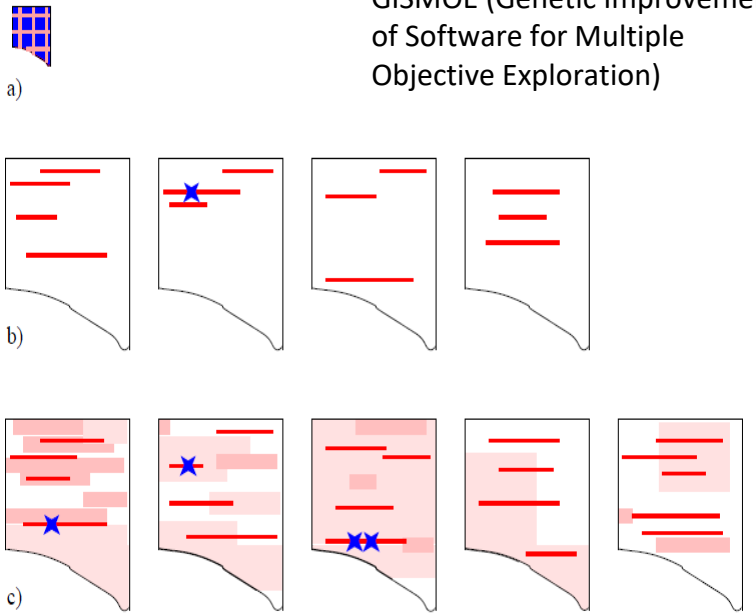
Each generation mutation and crossover create new modifications. Each modification's fitness is evaluated by applying it to the grammar and then reversing the grammar to get a new variant of the system.

Each modified system is tested on a randomised subset of the test suite and its answers and resource consumption compared to that of the original system. Modifications responsible for better systems procreate into the next generation.




GISMOE (Genetic Improvement of Software for Multiple Objective Exploration)

# Solution - GISMOE




GISMOE (Genetic Improvement of Software for Multiple Objective Exploration)

a) Initial approaches considered whole program equally (shaded). They update code (dark shading) which may be throughout the single source file.

b) Bug fixing. Genetic programming is directed to parts of code needing fixing (shaded) and the bugfix (star) is small.

c) GISMOE: Evolution is directed to used and heavily used code (light shaded, shaded, heavily shaded) several lines of code may be updated (stars).

# MOTIVATION - Bowtie2

**Why Bowtie2 as target system?**

**Ans:** Bowtie2 is a **complex bioinformatics tool used for aligning DNA sequences**, and it is widely used in genomics research. Bowtie2 system consists of **50 000 lines of C++ spread over 50 main system modules and 67 header files** (plus documentation, scripts and support modules).

Bowtie2 is considered an improved and more efficient version of its predecessor, Bowtie1. Bowtie2, like any software, has known issues, bugs, and performance bottlenecks.

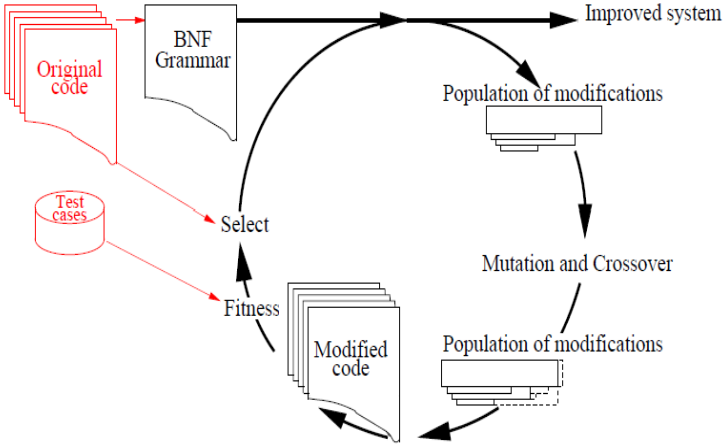
Genetic programming techniques provide an opportunity to automatically evolve improved versions of the software, addressing known issues and potentially discovering new ways to enhance its functionality.

# Implementation: GISMOE – for Bowtie2 program

**OVERVIEW:**

Various techniques and approaches are used to achieve genetic improvement in GISMOE:

- output binning
- semantic improvement
- sensitivity analysis
- operator choice
- grammatical representation
- local search.



# Implementation: GISMOE – for Bowtie2 program

**OUTPUT BINNING: O-Bins (Output Bins for Test Cases):**

- O-Bins are used to partition the available test cases. The motivation behind this partitioning is to capture the varying levels of difficulty associated with test cases.

- O-Bins allow the selection of test cases from different bins, ensuring a mix of difficult and less demanding tests for fitness evaluation. This helps in assessing both functional and non-functional properties of the code.

# Implementation: GISMOE – for Bowtie2 program

**SENSITIVITY ANALYSIS:**

- Sensitivity analysis is applied to identify the parts of the system that have the greatest impact on the non-functional property of interest (e.g., execution time).
- This analysis helps prioritize which parts of the system should be improved during the evolutionary phase.
- The weighting of lines of code takes into account:
  - how frequently they are executed and
  - how their use scales with the difficulty of the problem.

# Implementation: GISMOE – for Bowtie2 program

**REPRESENTATION OF THE SYSTEM TO BE EVOLVED:**

- **Template based Evolution:** The existing program serves as a template for its own improvement.
- A specialized "one-sentence" Backus-Naur Form (BNF) grammar is automatically created from the program's source code.


- Here Fundamental structures (classes, types, functions, & data structures) remain intact. Block structure is maintained i.e. the open/close brackets remain the same.

# Implementation: GISMOE – for Bowtie2 program

- **Code Compilation:** To ensure that evolved code is **syntactically valid**, a specialized BNF grammar is used **to eliminate parse errors**.

- **Reusing Code Fragments**: **Instead of** allowing GP complete freedom to **generate entirely new code**, the approach focuses on **reusing code** that has already been written within the program being improved. Human-written code often contains repeated lines or fragments. So, reusing existing code fragments yield significant improvements in the software.

# Implementation: GISMOE – for Bowtie2 program

- **Representation of a Genetically Improved Variant:** Each genetically improved variant is represented as an ordered list of changes to be made to the program's Backus-Naur Form (BNF) grammar. Changes include deleting lines of code, replacing lines with others, or inserting new lines.

- **Selection:** Up to half of the current population of genetic variants is selected based on their fitness (effectiveness). Variants that didn't compile or didn't surpass the original code's performance are excluded.

# Implementation: GISMOE – for Bowtie2 program

- **Mutation:** Genetic variants are mutated by appending new grammar modifications to their lists of changes. The line to be mutated is chosen based on sensitivity analysis and test case execution.

- **Crossover:** Crossover combines two genetic variants by concatenating their lists of changes.

- **Post Processing Solution Cleanup:** After genetic evolution, evolved programs can be bloated with unnecessary changes. A hill climbing strategy is used to minimize the size of the ordered list of changes.

# Evaluations GISMOE – for Bowtie2 program

- Consider a table with

-n entries, network with k nodes.

-The chromosome is coded as an array with n elements, each having a value between 1 and k-1. The *fitness* function has to include the following optimum criteria**: i) grouping of entries with the same attributes**, as well as **ii) distributing them in an equal fashion between a minimum number of nodes**.

$$GI = \sum_{i=1}^{k} \sum_{j=1}^{m} \left( \frac{A(i,j)}{n_i} \right)^2 \quad (2)$$

where:

| | |
|---|---|
| GI | – attribute groping |
| k | – number of distinct attributes |
| m | – number of nodes |
| A(i,j) | – number of attributes i from node j |
| $n_i$ | – total number of attributes i |

# Evaluations GISMOE – for Bowtie2 program

-Distribution of entries with the same attributes between a minimum number of nodes if done using the following formula:

$$DI = \sum_{i=1}^{k} \sum_{j=1}^{m} |A(i,j) - b| \quad (3)$$

where:
DI – distribution of entries
b – maximum number of entries that can be held in a node

- coefa, coefb are 2 constants; coefa, coefb are chosen based on the importance given to each optimum criterion.

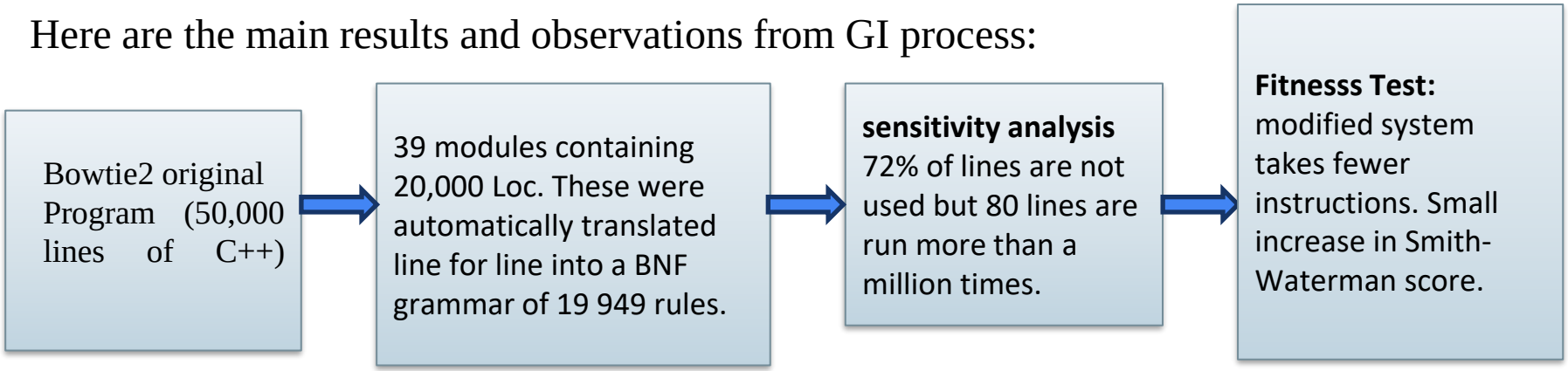
The fitness function is:

$$F = coefa\frac{GI}{k} + coefb\frac{1}{1+DI}, coefa + coefb = 1,\ coefa\ si\ coefb < 1 \quad (4)$$

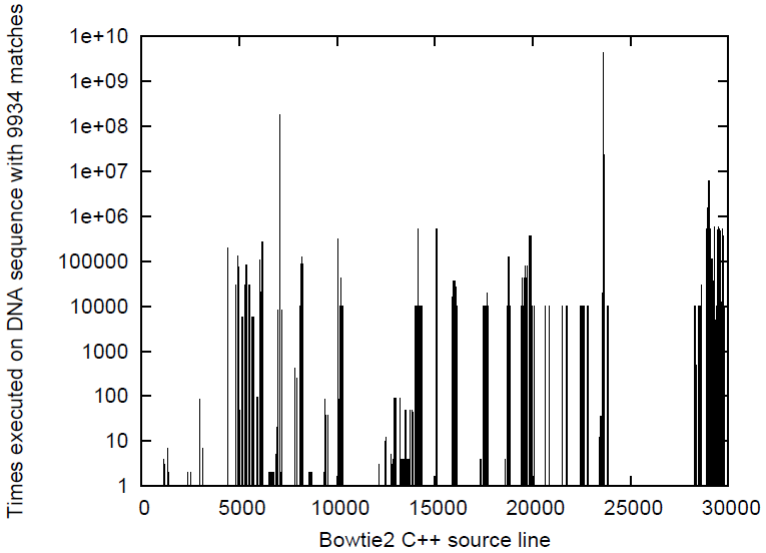
# Results - for Bowtie2 program

- The **Smith-Waterman algorithm** is used as an oracle to compare the results of Bowtie2 against the human genome and Solexa DNA sequences from the 1000 Genomes Project.

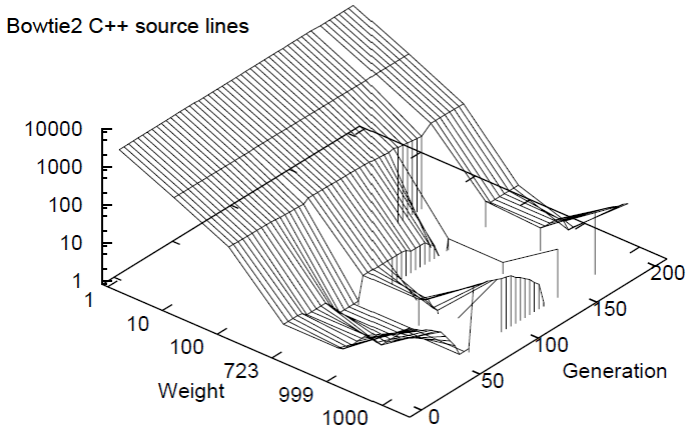
Here are the main results and observations from GI process:



# Results - for Bowtie2 program

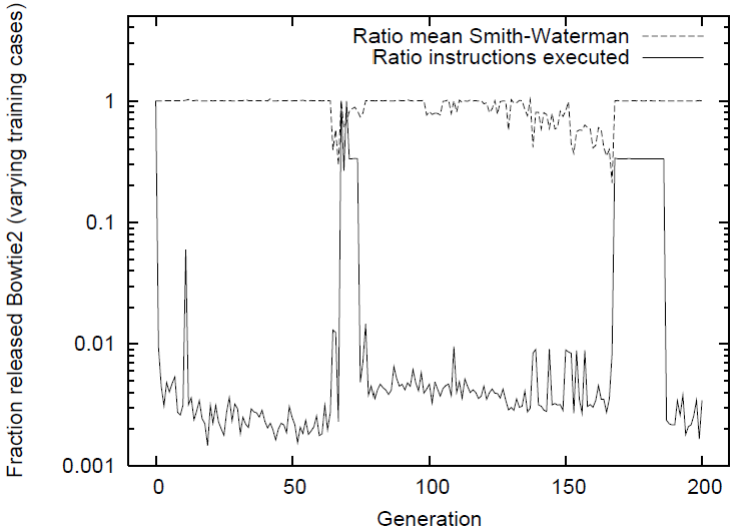



When Bowtie2 finds many matches of the distribution of the number of times each Bowtie2 C++ source line is used. **72% of lines are not used but 80 lines are run more than a million times**




Evolution of distribution of weights. Initial GP population **(generation 0)** at left. Note fall in proportion of higher weights as population evolves to mutate highly used C++ lines and then recovery as it (partially) resets in generations **68 and 168**. (Note non-linear scales. To reduce clutter, data plotted only every **4th generation**.)

# Results - for Bowtie2 program




**Evolution of performance:** In the last generation the best uses 290 times fewer C++ statements than the original code. In most generations there is a small improvement in mean Smith-Waterman score (dashed line)

# Results - for Bowtie2 program

- **Performance:** The evolved Bowtie2 version is tested on a hold-out dataset, and the results show that it is on average **74 times faster than the original version**.

| Weight | Mutation | Source file | line | type | Original Code | New Code |
|---|---|---|---|---|---|---|
| 999 | replaced | bt2_io.cpp | 622 | for2 | `i < offsLenSampled` | `i < this->_nPat` |
| 1000 | replaced | sa_rescomb.cpp | 50 | for2 | `i < satup_->offs.size()` | `0` |
| 1000 | disabled | sa_rescomb.cpp | 69 | for2 | `j < satup_->offs.size()` | |
| 100 | replaced | aligner_swsse_ee_u8.cpp | 707 | | `vh = _mm_max_epu8(vh, vf);` | `vmax = vlo;` |
| 1000 | deleted | aligner_swsse_ee_u8.cpp | 766 | | `pvFStore += 4;` | |
| 1000 | replaced | aligner_swsse_ee_u8.cpp | 772 | | `_mm_store_si128(pvHStore, vh);` | `vh = _mm_max_epu8(vh, vf);` |
| 1000 | deleted | aligner_swsse_ee_u8.cpp | 778 | | `ve = _mm_max_epu8(ve, vh);` | |

# Critique

**Soundness: do you see any flaws in the paper?**

- The paper demonstrates soundness in its approach to applying genetic programming (GP) to evolve and optimize software. It follows a systematic process **and provides a clear methodology for evolving code**.
- The **paper could provide more insights into how the choice of parameters or hyperparameters** in the GP process might affect the results and whether sensitivity analysis was performed.

**Significance: do you think that the approach could be applied in practice?**

- The paper's contributions are significant for the field of software engineering. It introduces the idea of using GP to evolve **and optimize existing software**, **potentially improving both functional and non-functional properties.**
- This approach can be further improvised and **can be used for many real-life complex systems.**

# Critique

**Novelty: what are the main novelty aspects of the paper?**

- There have been other similar  works related to Genetic Improvement using GISMOE method. But this paper uses the Bowtie2 system to achieve the GI which is indeed novel.

**Verifiability and Transparency: Do you think the work presented in the paper is reproducible?**

- Access to the codebase used in the experiments are publicly available. So, it greatly enhances the transparency and verifiability of the research.

**Presentation:** The paper is easy to read and follow. (Personal opinion: Quiet a lengthy paper)

# Conclusion

The genetic improvement approach successfully enhances/enhanced the performance of Bowtie2 system, making it more efficient for DNA sequence alignment without sacrificing functionality.

# References

[1] Mark Harman, William B. Langdon, Yue Jia, David R. White, Andrea Arcuri, and John A. Clark, "The GISMOE challenge: Constructing the Pareto program surface using genetic programming to find better programs," in The 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 12), Essen, Germany, Sept. 3-7 2012, pp. 1–14, ACM.

[2] W. B. Langdon and M. Harman, "Evolving a CUDA kernel from an nVidia template," in 2010 IEEE World Congress on Computational Intelligence, Pilar Sobrevilla, Ed., Barcelona, 18-23 July 2010, pp. 2376– 2383, IEEE.

[3] Michael Orlov and Moshe Sipper, "Flight of the FINCH through the Java wilderness," IEEE Transactions on Evolutionary Computation, vol. 15, no. 2, pp. 166–182, Apr. 2011.

[4] David R. White, Andrea Arcuri, and John A. Clark, "Evolutionary improvement of programs," IEEE Transactions on Evolutionary Computation, vol. 15, no. 4, pp. 515–538, Aug. 2011.

[5] Andrea Arcuri and Xin Yao, "A novel co-evolutionary approach to automatic software bug fixing," in 2008 IEEE World Congress on Computational Intelligence, Jun Wang, Ed., Hong Kong, 1-6 June, pp. 162–168.

[6] Cristian Cadar, Peter Pietzuch, and Alexander L. Wolf, "Multiplicity computing: a vision of software engineering for next-generation computing platform applications," in Proceedings of the FSE/SDP workshop on Future of software engineering research, Kevin Sullivan, Ed., Santa Fe, New Mexico, USA, 7-11 Nov. 2010, FoSER '10, pp. 81–86, ACM.

[7] Ben Langmead and Steven L Salzberg, "Fast gapped-read alignment with Bowtie 2," Nature Methods, vol. 9, no. 4, pp. 357–359, 2012.

[8] Richard M. Durbin, et al., "A map of human genome variation from population-scale sequencing," Nature, vol. 467, no. 7319, pp. 1061– 1073, 28 Oct 2010.

# Thank You
# Any Questions?