CSE/ECE 848 Introduction to Evolutionary Computation

Module 5 - Lecture 23 - Part 1b (formerly 3)

Genetic Improvement

Wolfgang Banzhaf, CSE
John R. Koza Chair in Genetic Programming

Outline

- Genetic Improvement (GI)
- ARJA: GI for Repairing Bugs in Java

Genetic Improvement - The Idea

- Genetic Improvement (GI) is related to the original goal of GP (i.e., automatic programming), rather than symbolic regression.
- GP generally cannot start from scratch to create a real-world program
- GI offers a different slant: why not take human-written program as a starting point and improve it

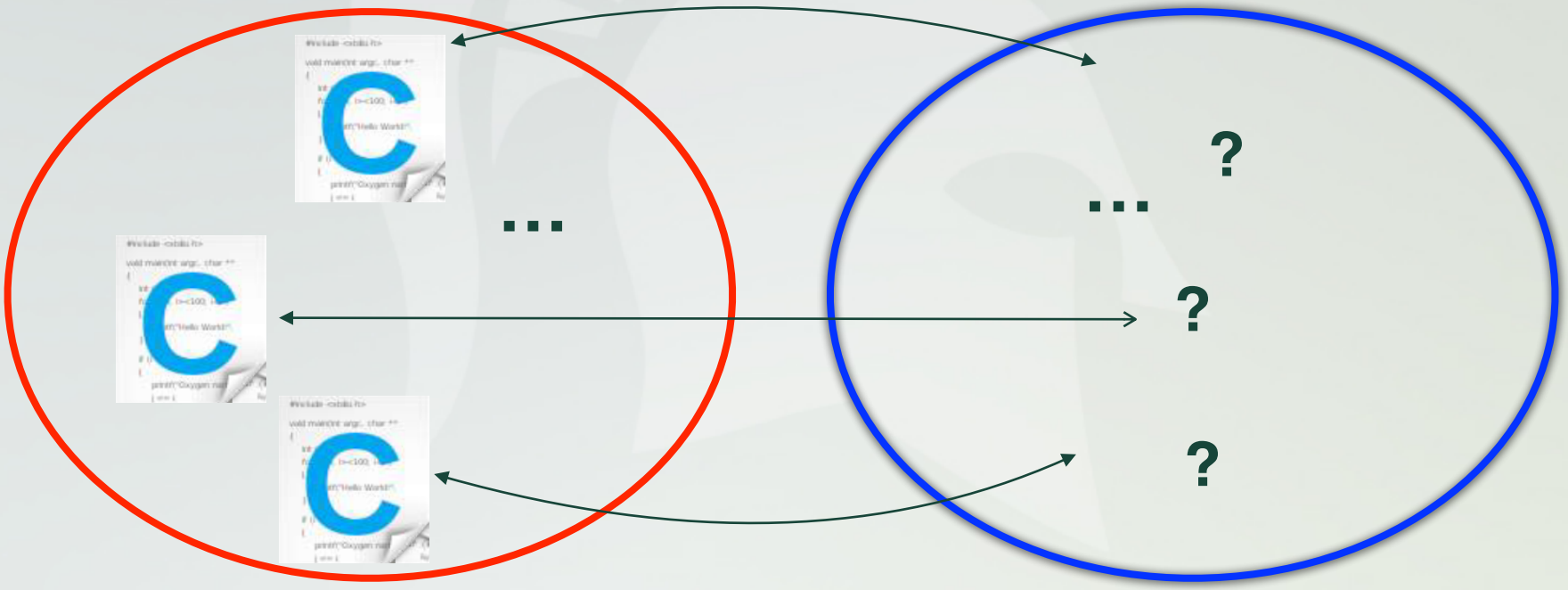
Genetic Improvement - Types

- There are two types of improvements
  - Functional properties: from a buggy program to a correct program (bug repair)
  - Non-Functional properties: e.g., from a slow program to a faster program (software optimization)
- GI = Software improvement through GP

Genetic Improvement

Original software

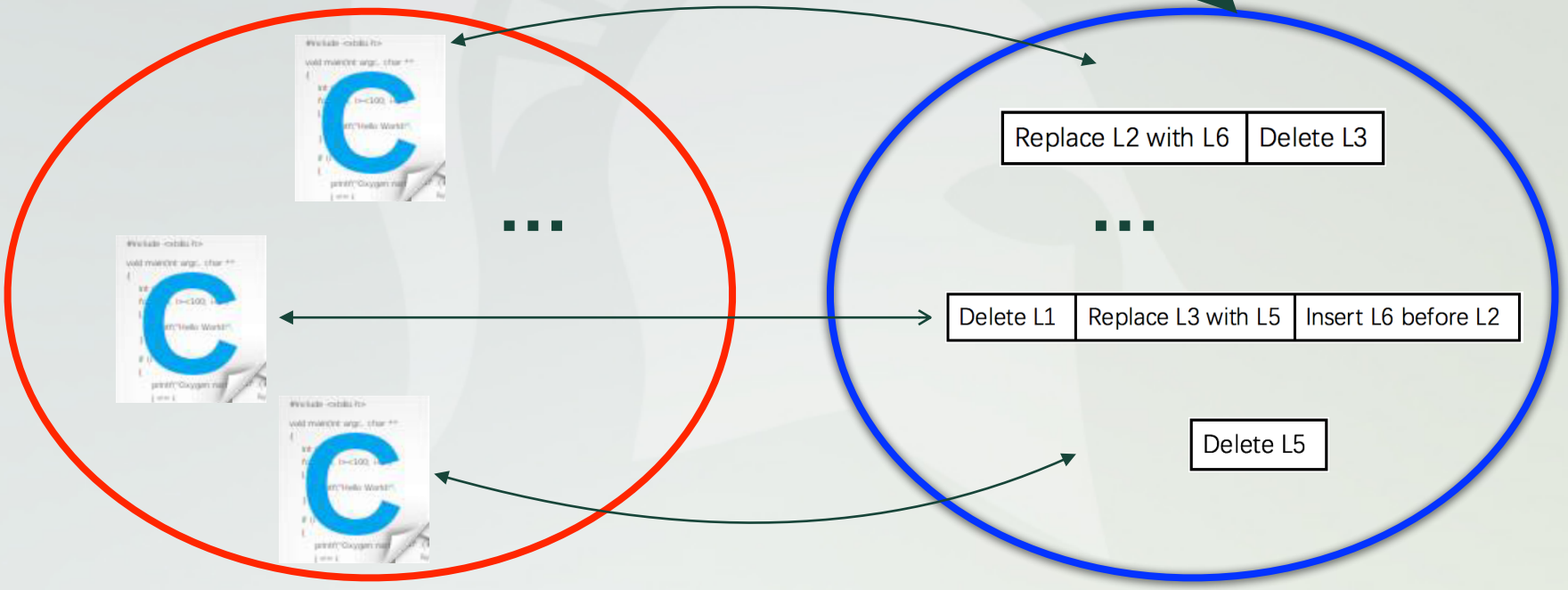
Here each solution is really a (large-scale) software, but it is *very similar* to the original software



Genetic Improvement - Patches

Original software

Each genome is a list of modifications to the original software, or called patch

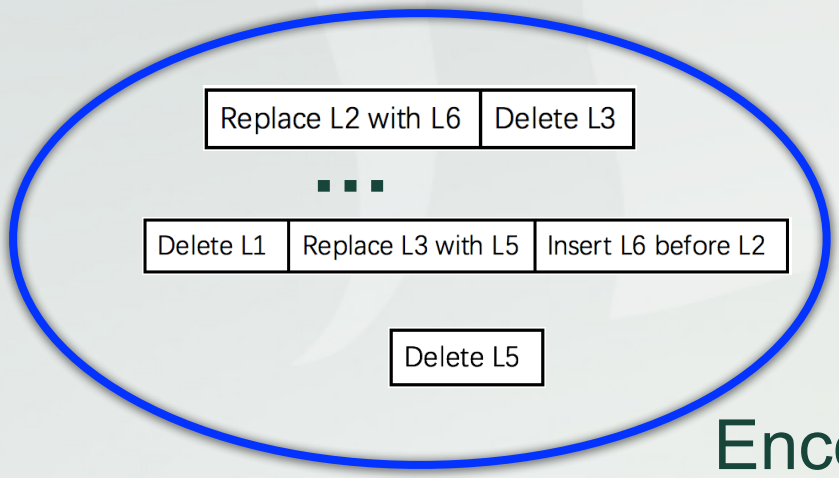


Solution space

Encoded space

Genetic Improvement - How?

- AST tree structure does not scale to large software
- A population of AST trees for large software cannot fit in computer memory
- The trees are normally very similar to the tree of the original software
- Representing each genome as a list of modifications can avoid storing redundant copies of untouched code



Why not use the AST tree of the software?

Encoded space

Genetic Improvement - Bug Repair

- GI for bug repair: given a buggy program and a set of test cases (with at least one test case failing), conduct a biased GP search to improve the buggy software, in order to make all the test cases passing.

An initially failing test case (negative test case)

Input	Output
0	1
2	2
3	6
5	120
… ….	

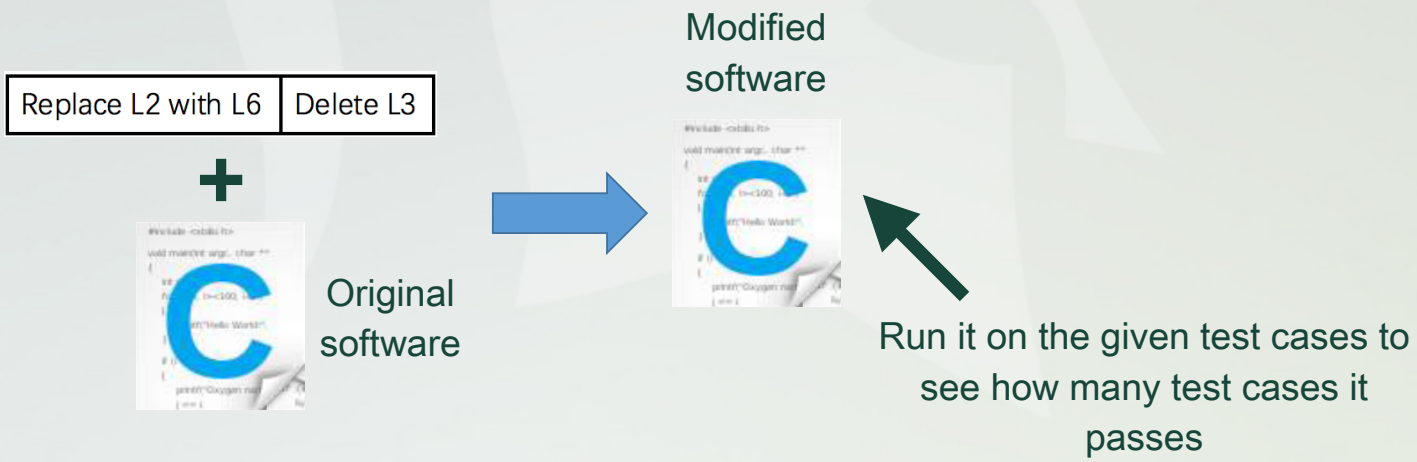
A set of test cases

```
int Factorial(int a) {
    if (a <= 0)
        return 0;
    else
        return (a * Factorial(a - 1));
}
```

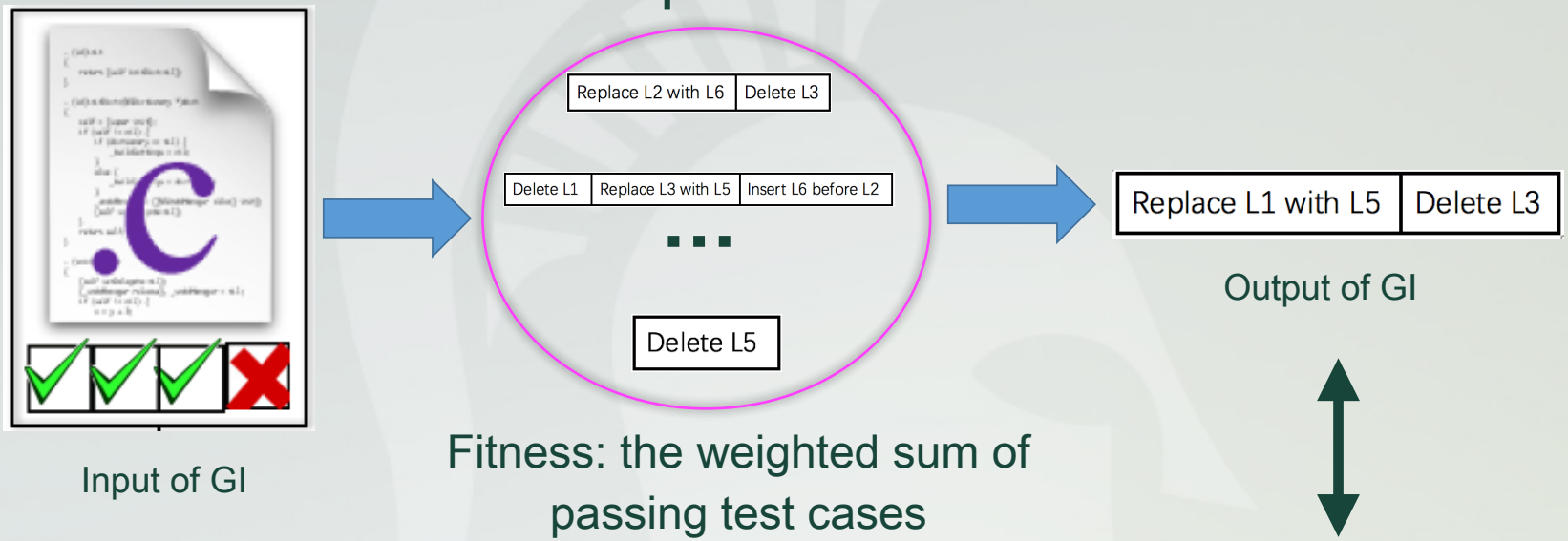
A buggy program

Genetic Improvement - Bug Repair II

- Fitness function (GI for bug repair)
  - Give a genome/patch (i.e., a list of modifications), how to compute the its fitness
  - Fitness: the weighted sum of the number of test cases that are passed



Genetic Improvement

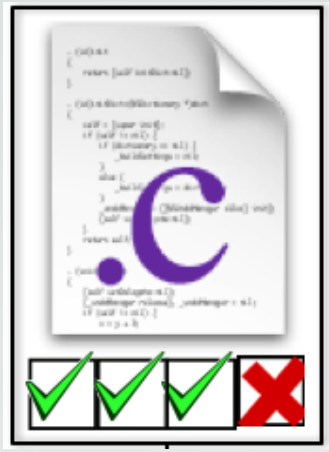


Overall procedure of GI for bug repair

A repaired software

Genetic Improvement

- Search space of patches in GI for bug repair
  - Where (e.g., which lines of code) we want to modify?



A buggy software with test cases

Fault localization tool

Likely Buggy Lines
L12
L20
L87
…
L201

The lines we may change

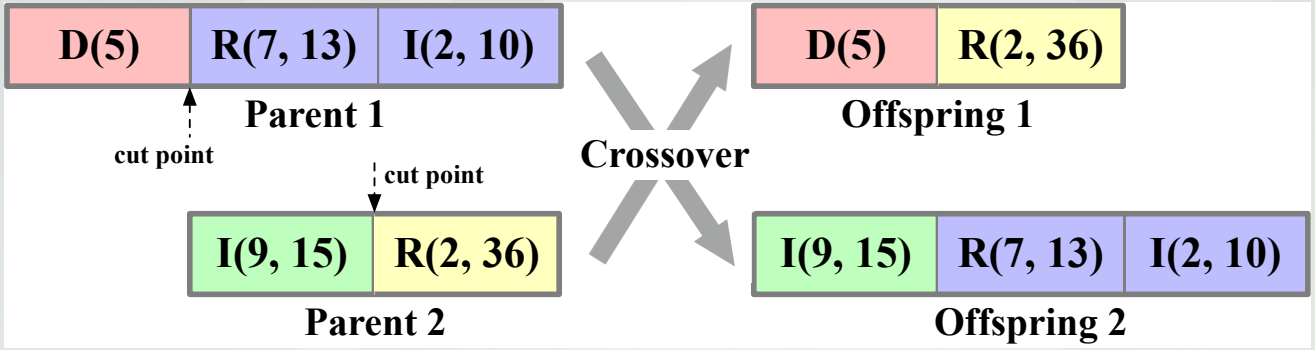
Genetic Improvement

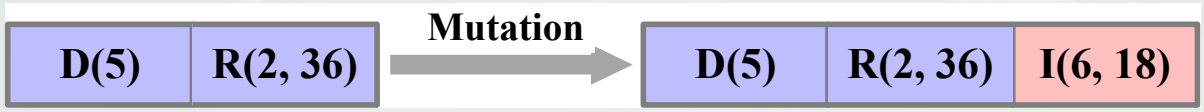
- Search space of patches in GI for bug repair
  - Which kind of modifications (of a line) we want to conduct?
    A. Delete a line
    B. Replace a line with another code
    C. Inert another code before a line
  - Another code?: assume that it just comes from the code in the current buggy software
  - Redundancy assumption: we can usually fix a bug only using the existing code in the buggy software

Le Goues, C., Nguyen, T., Forrest, S., & Weimer, W. (2011). GenProg: A generic method for automatic software repair. *IEEE Transactions on Software Engineering*, 38(1), 54-72.

Genetic Improvement

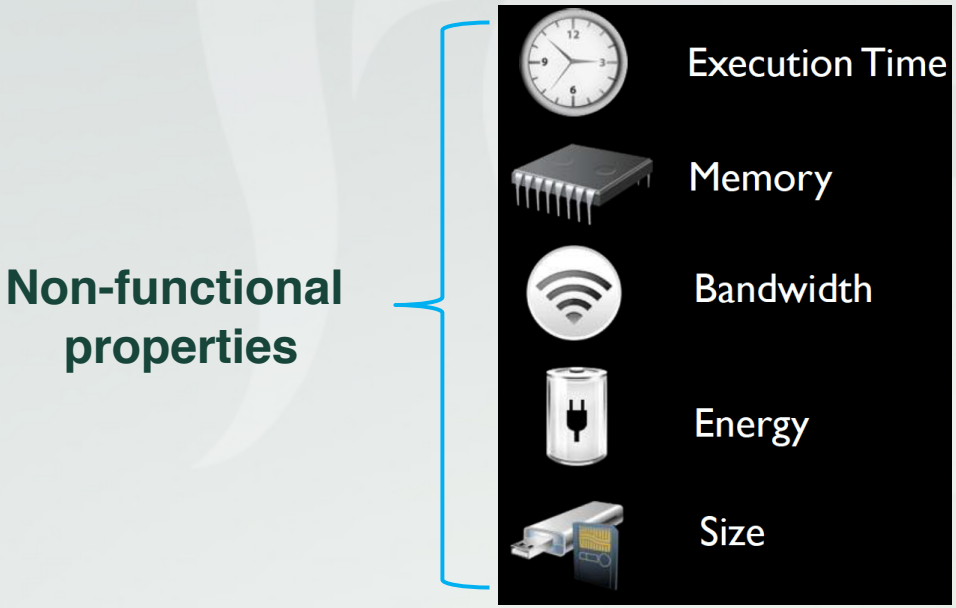
- Genetic Operators (GI for bug repair)
  - Crossover and mutation



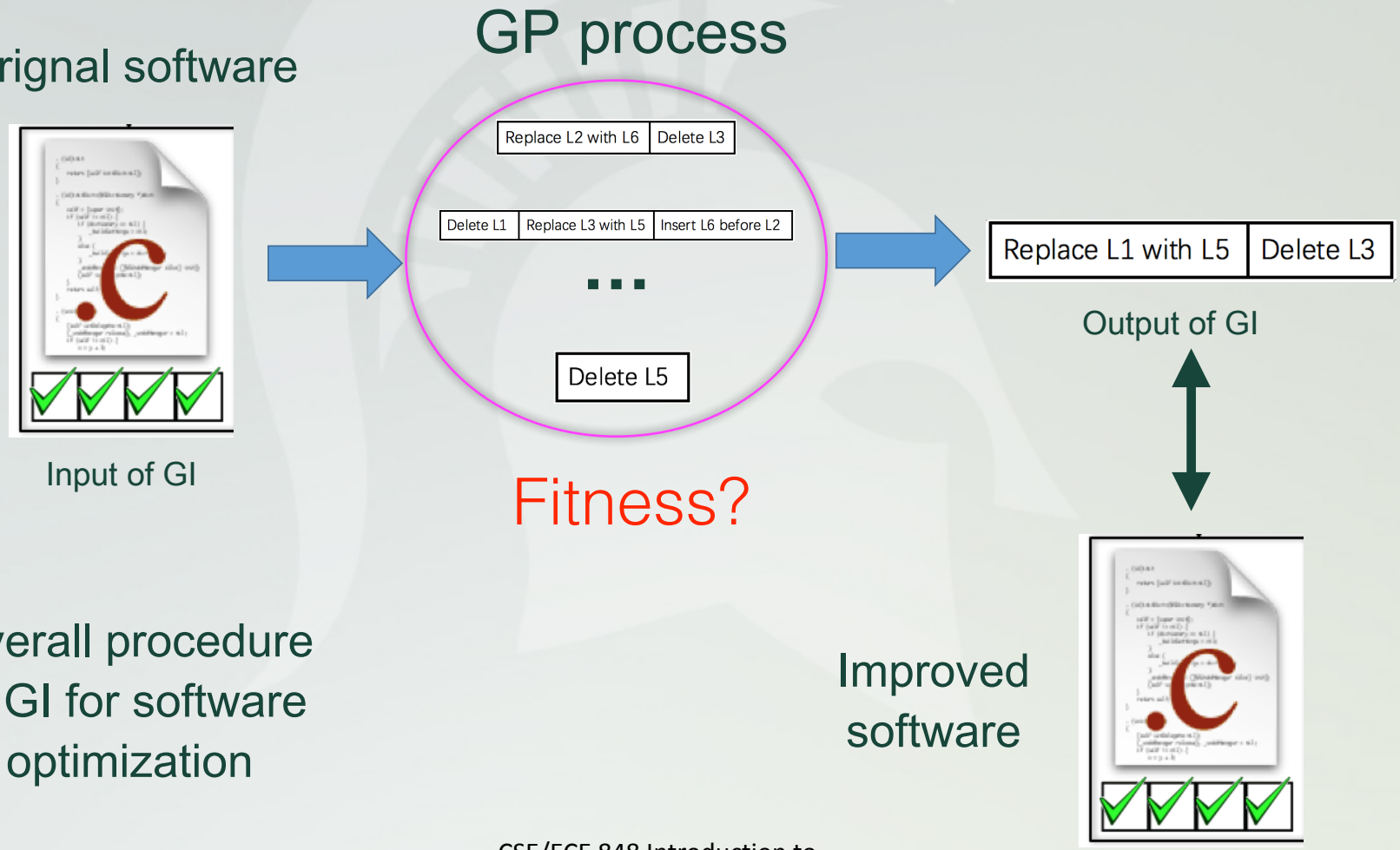


Genetic Improvement - Optimization

- GI for Software Optimization: improve non-functional properties (e.g., running time, memory consumption, power/energy consumption) of a software



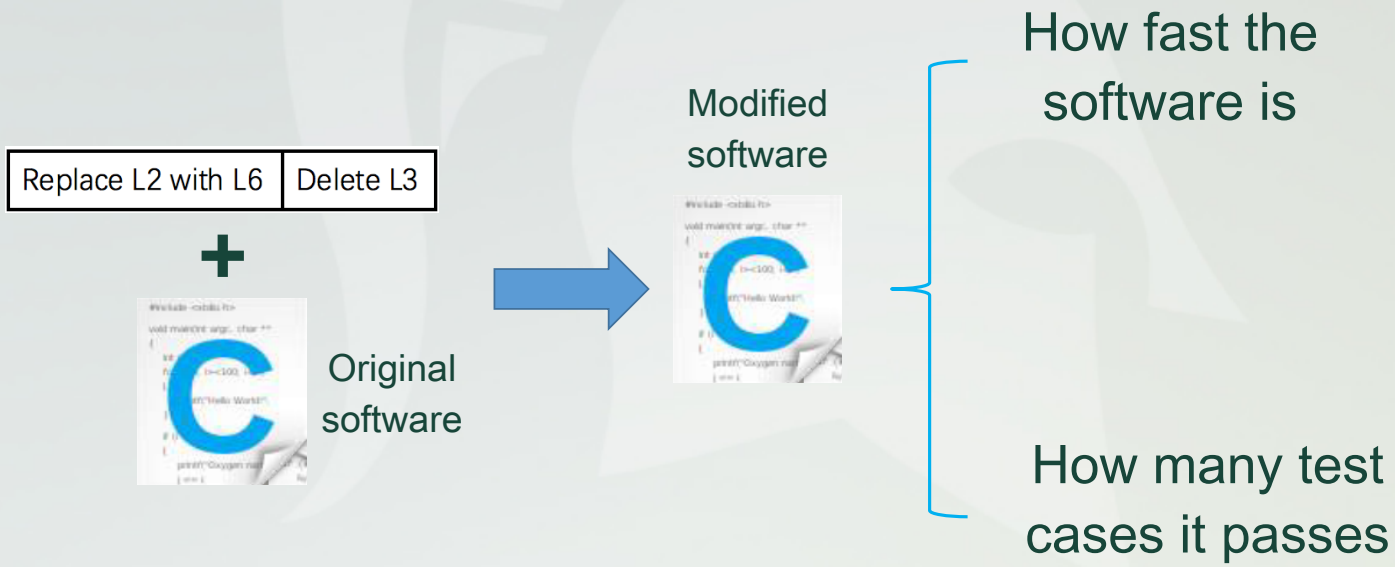
Genetic Improvement



Overall procedure of GI for software optimization

Genetic Improvement

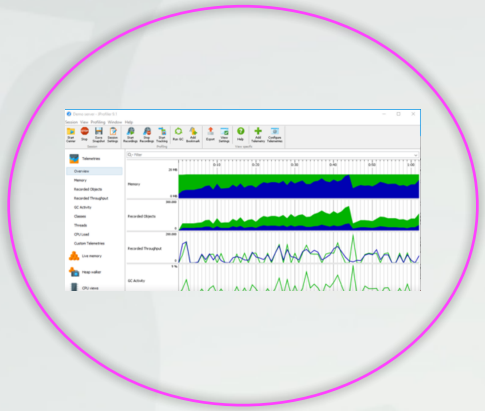
- GI for software optimization: fitness function



Genetic Improvement

Original software

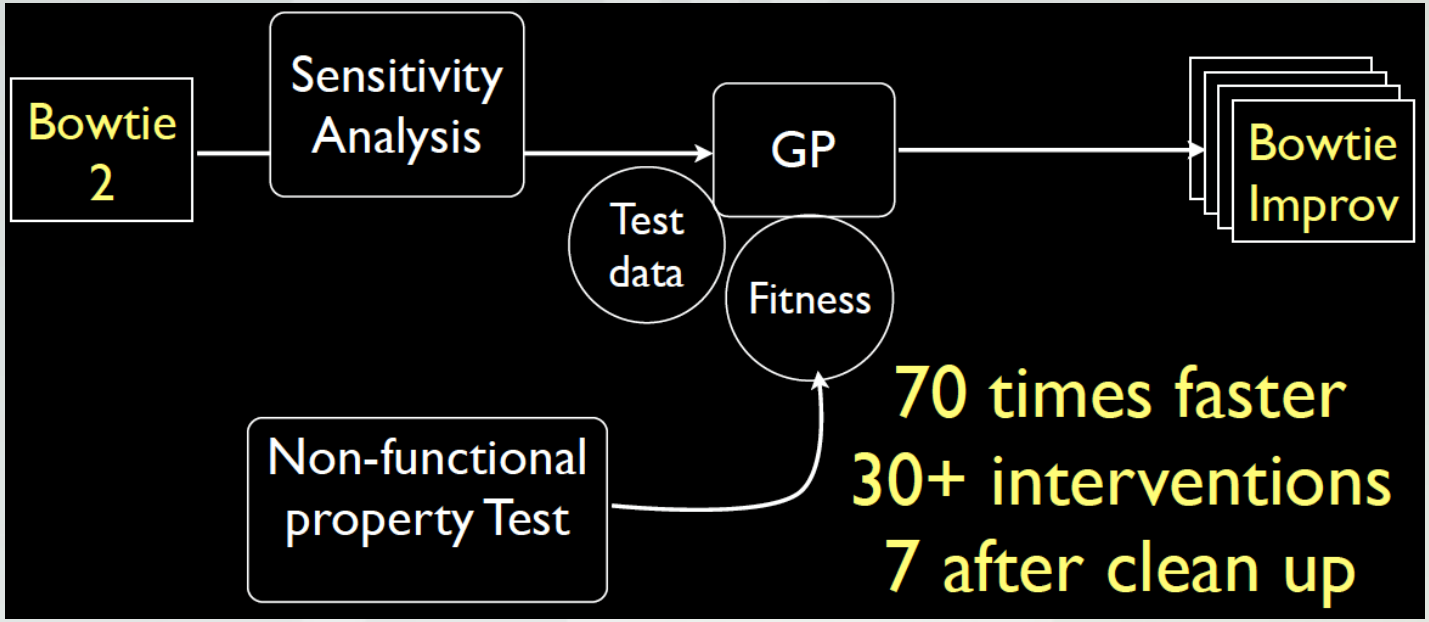
Performance profiling tool



Bottleneck lines
L12
L20
L87
…
L201

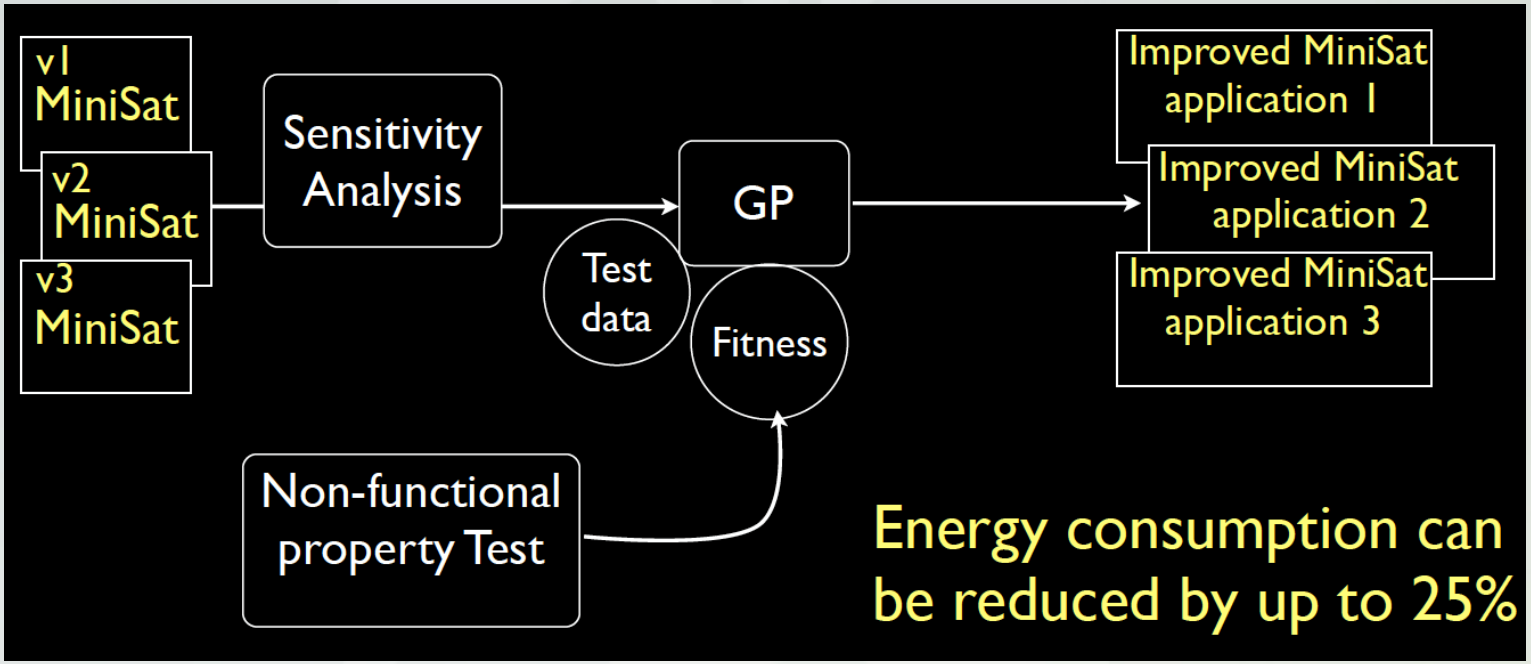
The lines we may change

Genetic Improvement - Application



William B. Langdon and Mark Harman, Optimizing Existing Programs with Genetic Programming, IEEE Transactions on Evolutionary Computation, 19 (1), 2015.

Genetic Improvement



Bobby R. Bruce, Justyna Petke, Mark Harman, and Earl T. Barr, Approximate Oracles and Synergy in Software Energy Search Spaces, IEEE Transactions on Software Engineering, in press

Genetic Improvement - A Problem

- Overfitting in GP/GI
  - The output program/software may be incorrect beyond passing the given test cases
  - The underlying reason is that the test suite is usually an incomplete program specification

Input	Output
0	1
2	2
3	6
5	120
6	720

Test cases

```
int Factorial(int a)
{
  if (a <= 0)
     return 1;
  else
     return (a * Factorial(a-1));
}
```

Correct program (very likely)

```
int Factorial(int a) {
    switch (a) {
    case 0: return 1;
    case 2: return 2;
    case 3: return 6;
    case 5: return 120;
    case 6: return 720;
    default: return 0;
    }
}
```

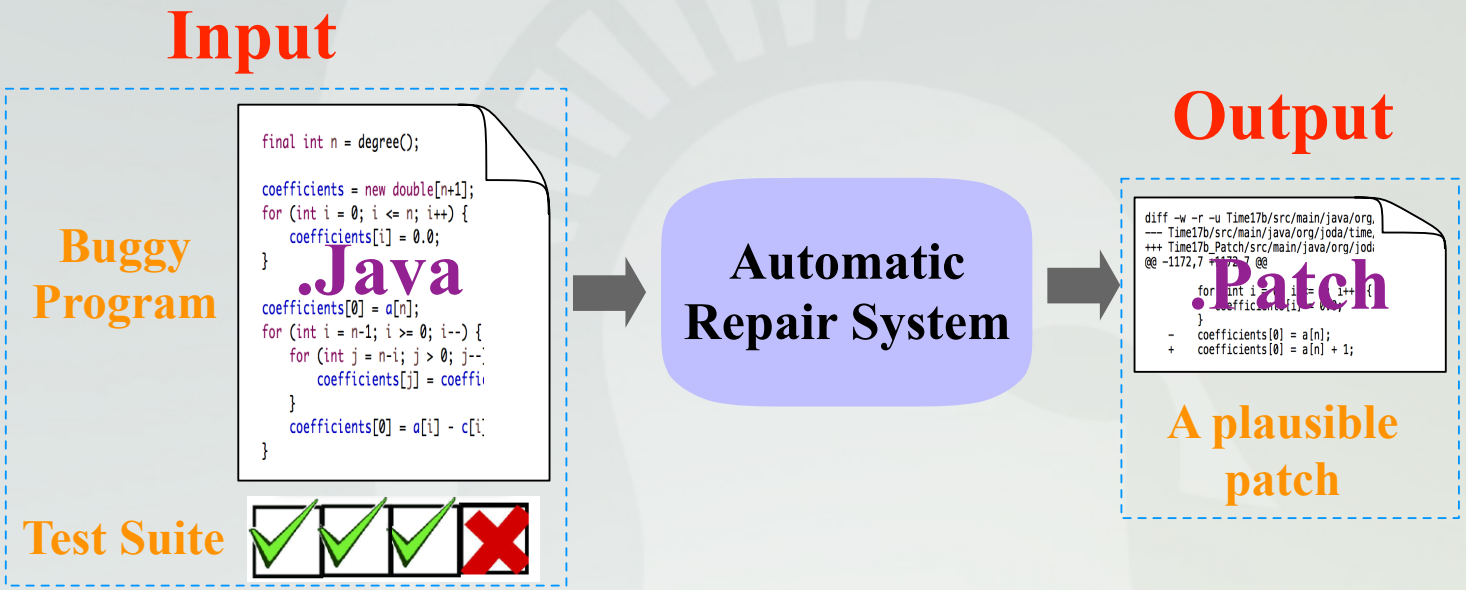
Overfitting!

Genetic Improvement

- Overfitting in GP/GI
  - This is a very tricky problem
  - Usually, we use GP/GI to output a number of programs, and then use certain kind of heuristic to rank the output programs (ranked higher, more likely to be correct).

ARJA: GI for Repairing Bugs in Java

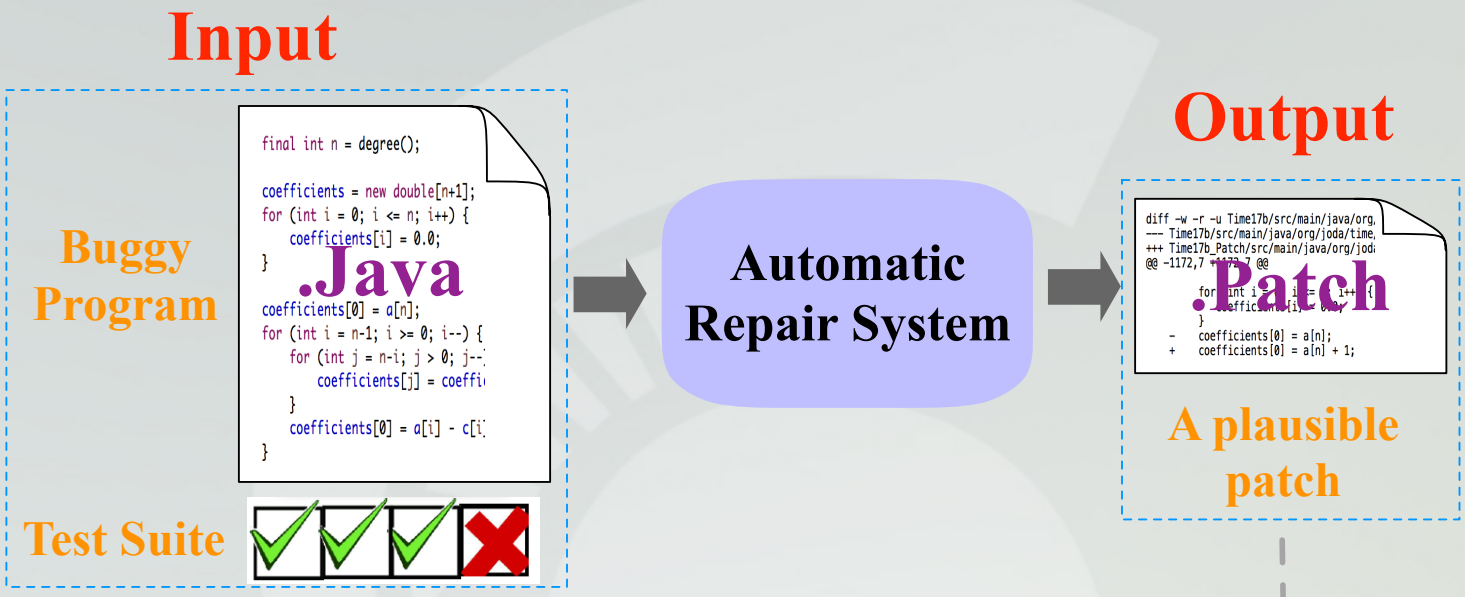
- Problem Statement (Test-Suite Based Bug Repair)



Note:

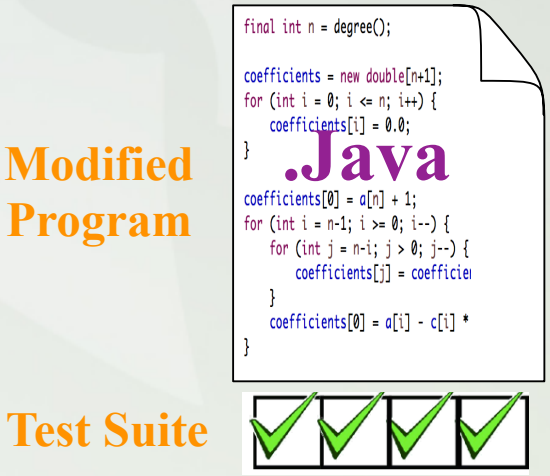
1. Test suite should contain a number of initially passing (i.e., positive) test cases and at least one initially failing (i.e., negative) test case
2. Sometimes the number of plausible patches obtained can be more than one.

Problem Statement



- A plausible patch: a patch that can make the entire test suite pass.

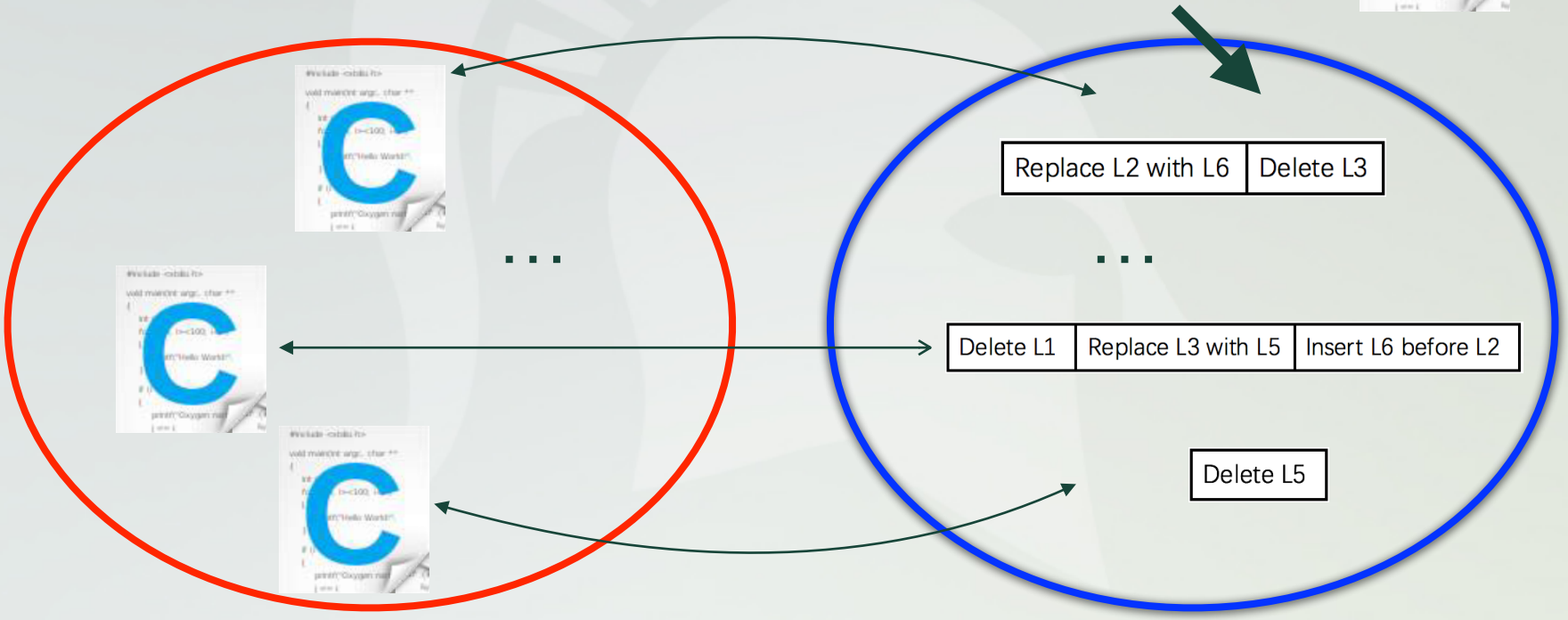
Note: A plausible patch is a patch that can make the modified program pass the entire test suite.



ARJA: GI for Repairing Bugs in Java

Original software

Each genome is a list of modifications to the original software, or called patch



ARJA: GI for Repairing Bugs in Java

- Components in a GI System for Bug Repair
  - Search space (which patches we want to consider)
  - Search algorithm
    - Fitness function (how to define the goodness of a patch)
    - Find best patches using GP (over a genetic representation)
  - Handling the overfitting issue

ARJA: GI for Repairing Bugs in Java

- A detailed introduction can be found in our papers:

[1] Yuan Yuan, Wolfgang Banzhaf, ARJA: Automated repair of Java programs via multi-objective genetic programming. *IEEE Transactions on Software Engineering, Vol 46, 1040—1067 (2020)* [This paper introduces the original ARJA system]

[2] Yuan Yuan, Wolfgang Banzhaf. Towards better evolutionary program repair: An integrated approach. *ACM Transactions on Software Engineering and Methodology, Vol 29, 5:1—5:53 (2020)* [This paper introduces an enhanced ARJA version called ARJA-e]