Genetic improvement: Taking real-world source code and improving it using genetic programming

Sæmundur Ó. Haraldsson, John R. Woodward, Markus Wagner

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

GECCO '20 Companion, July 8–12, 2020, Cancún, Mexico
© 2020 Copyright is held by the owner/author(s).
ACM ISBN 978-1-4503-7127-8/20/07
https://doi.org/10.1145/3377929.3389885

GECCO 2020

Overview

- Introduction
- Fixing Bugs and other examples
- Noteworthy papers and issues
- Getting involved
- Summary and Q&A
What is Genetic Improvement

A wordy definition:
Genetic Improvement is the application of search-based (typically evolutionary) techniques to modify software with respect to some user-defined fitness measure.

It's just GP - BUT starting with a nearly complete program [Wolfgang Banzhaf]

What is Genetic Improvement

GI

Improve Functional Properties

Automatic Bug fixing

Functional Properties

Feature Transplantation

Grow and Graft

Auto-parallelisation

Improve non-functional properties

Improve memory consumption

Software Slimming

GI

Improve execution time

Improve energy consumption

Software Slimming

Auto-parallelisation

Genetic Programming overview

Aim – to discover new programs by telling the computer what we want it to do, but not how we want it to do it – John Koza

How – we evolve computer programs using natural selection.

Starts from scratch (empty program)

Choose primitives (terminal set/FEATURES and function set)

Choose representation (tree based, graph based, linear e.g. CGP)

Choose fitness function, parameters, genetic operators.

Genetic Programming: GI’s ROOTS
GI forces “the full capabilities of programming languages” - side effects, ADFs, LOOPS

GP vs GI: if you can’t beat them, join them.

John R. Woodward
University of Stirling
Stirling, Scotland, United Kingdom
jrw@cs.stir.ac.uk

Colin G. Johnson
University of Kent
Kent, England, United Kingdom
C.G.Johnson@kent.ac.uk

Alexander E. I. Brownlee
University of Stirling
Stirling, Scotland, United Kingdom
sbr@cs.stir.ac.uk

ABSTRACT
Genetic Programming (GP) has been criticized for target-
ing irrelevant problems [12], and in terms of the wider machine
(procedures, methods, macros, routines), and so GI has to
deal with the reality of existing software systems. How-
ever, most of the GP literature is not concerned with Tran-
Competent Programmers Hypothesis

1. Programmers write programs that are almost perfect.
2. Program faults are syntactically small (slip of finger, T/F)
3. Corrected with a few keystrokes. (e.g. < for <=)
4. GI can find small patches.
5. Small changes are non-unique (7 lines code, or utter 7 words before unique)
Plastic Surgery Hypothesis.

the content of new code can often be assembled out of fragments of code that already exist.

Barr et al. [71] showed that changes are 43% graftable from the exact version of the software being changed.

The Plastic Surgery Hypothesis: Changes to a codebase contain snippets that already exist in the codebase at the time of the change, and these snippets can be efficiently found and exploited.

THE CODE CONTAINS SOLUTIONS – CANDIDATE PATCHES

Objectives

- Functional (logical properties)
  - Accuracy e.g. as in machine learning - FLOAT
  - Number of bugs – as measured against a set of test cases. BOOLEAN
  - New functionality – e.g.
- Non-functional (physical properties)
  - Execution time
  - Energy (power consumption – peak/average)
  - Memory
  - Bandwidth
- Multi-objective
  - Trade-offs, convex, a set of programs = a single tuneable program

Multi-Objective

- Seems be convex
- – simple argument (see pic)
- Can provide a set of programs
- weighted sum of objectives?
- weight have meaning to user.
- Will there be elbow/knee points?

Representations of PROGRAMS

Natural Representation of CODE

1. Text files e.g. Program.java is a text file. Saemi.
3. Java byte code (also C binaries) [102]
4. Errors, compile, halting (Langdon - discard)
The GISMOE challenge:
to create an automated program development environment in which the Pareto program surface is automatically constructed to support dialog with and decision making by the software designer concerning the trade offs present in the solution space of programs for a specific programming problem.

EDITS Operators – changes to programs

- Line level
- Single Character level
- Function/module level.
- AST – GIN, Gen-0-fix, genprog.
- Java – machine code – java byte code.

- LIST OF EDITS IS A PATCH.
GI: An example of automated bug fixing

```scala
class Example {
  def main(args: Array[String]): Unit = {
    val a = 1
    val b = 2
    val c = 3
    if (a + b < c)
      println("INVALID")
    else if (a == b && b == c)
      println("ISOCELES")
    else if (a == b || b == c)
      println("EQUALATERAL")
    else
      println("SCALINE")
  }
}
```

Main features of framework are

1. **Embedded** adaptively.
2. Minimal end-user requirements.
3. **Source to source transformations**
4. Operates on **ASTs** (i.e. arbitrarily fine).

---

**Gen-O-Fix: Abstract Syntax Trees**

Main features of framework are

1. **Embedded** adaptively.
2. Minimal end-user requirements.
   1. Initial source code: **location** of Scala source code file containing a function
   2. Fitness function: providing a means of **evaluating** the quality of system
3. **Source to source transformations**
4. Operates on ASTs (i.e. arbitrarily fine).
**Hadoop** provides a mapReduce implementation in Java.

1. **Hadoop** provides a mapReduce implementation in Java.
2. Equals method has to obey **contract** (Reflective, Symmetric, Transitive, ...)
3. x.equals(y) implies hashCode(x) == hashCode(y).
4. hashCode method is an integer function of a subset of an object’s fields.

---

**AST - scala**

Code as data, data as code.

```scala
// code to data:
var m = 2; var x = 3; var c = 4
val expr = reify((m * x) + c)
println("AST = " + showRaw(expr.tree))

// output:
AST = Apply(Apply(SELECT(SELECT(Ident("m"), "elem"), "times"), List(SELECT(Ident("x")), "elem"))), "plus") , List(SELECT(Ident("c"), "elem")))
```
Some GP Settings

1. **Terminal set** is
   1. Field values
   2. Random integers [0, 100]

2. **Function set** is
   1. \(+, *, \text{XOR}, \text{AND}\)

3. **Fitness function**: close to uniform distribution (uniform distribution is the ideal), over 10,000 instances.

Overview

- Introduction
- **Fixing Bugs and other examples**
  - Noteworthy papers and issues
  - Getting involved
  - Summary and Q&A

Fixing Bugs and other examples

Saemundur O. Haraldsson

- Fixing bugs
- Making software faster
- Making software more accurate
Fixing bugs

A real world example of GI in action


Janus Manager

- Management system for rehabilitation
- Web application
  - Python source code
  - >25K LOC
- ~200 users
  - ~40 specialists
  - 150-160 patients
- In use since March 2016
- 47 bugs automatically fixed to date

When last user logs out

1. Procedure 2.0
   - Sorts and filters the day’s exceptions

When last user logs out

1. Procedure 2.0 started
   - Sorts and filters the day’s exceptions
2. Procedure 3.0
   - Emulates input data, type, size and structure.
   - Produces test cases
When last user logs out

1. Procedure 2.0 started
   ● Sorts and filters the day’s exceptions
2. Procedure 3.0
   ● Emulates input data, type, size and structure.
   ● Produces test cases
3. Procedure 4.0
   ● Genetic Improvement
   ● Parallel process on the server
   ● Outputs report for developer
- **Procedure 4.0**
  - Genetic Improvement
  - Pop. = 50 patches
  - fit. = # passed tests
  - select = ½ pop by fitness
  - Output = report

**4 different types of implemented Edits**

**Primitive types:**
- **Copy**
  - Equivalent to CTRL+C → CTRL+V
- **Delete**
  - Almost what you think

**Composite types:**
- **Replace**
  - Copy + Delete
- **Swap**
  - 2x Copy + 2x Delete

**Copy**
- CTRL+C => CTRL+V
- Applied to whole lines
- Some restrictions on what lines can be copied
- Identified with regular expressions

**Delete**
- Adds “#” to beginning of line
- “Comment”
- Applied to whole lines
- Some restrictions on what lines can be commented out
- Identified with regular expressions
- Can be reversed for previously deleted lines
- “Uncomment”
**Swap**
- Copies both lines above each other
- Then deletes the originals
- Applied to whole lines
- Like for like

**Replace**
- Copies one line above another
- Then deletes that line

**Replace -- extra**
- Deep parameter tuning
- Operator specific replacement
- and numbers too
- From a list of equivalent operators.

**A list of edits makes a suggestion**
- Reads like a recipe
- Step-by-step
- Automatically reduced
- Delta debugging
- Scrutinised by the developer
- Might change the recipe
A list of edits makes a suggestion

- Reads like a recipe
- Step-by-step
- Automatically reduced
- Delta debugging
- Scrutinised by the developer
- Might change the recipe
Summary

- Real-world example
- Catches inputs that produce crashes
- Line(-ish) based GI
- 4 types of edits
- Overnight repair
- Developers are the gatekeepers

Faster

Another example of GI in action


The software

ProbABEL
- A tool for Genome Wide Association studies.
- Collection of functions for regression models
- Written in C and C++
  - 8k LOC
  - 31 files
- Typical execution time around 8-12 hours

The GI setup
- Same as before
- Except for the evaluation
- Mean CPU time from 20 executions
- None compiling and failing variants are not discarded

http://www.genabel.org/packages/ProbABEL
Results
- 2 good variants found early on
  - < a second faster
  - Generations 5 and 10
- Not statistically significant on training dataset

Variant 1
- Deletes a single line that performs an expensive matrix multiplication

Variant 2
- Changes: `i++` to `++i`

Better predictions
And even more examples of GI in action

---


Dynamic updates to a prediction tool

- Used by Janus Rehabilitation
  - Since June 2016
  - Consulted in all team meetings
- Updated whenever there are new information
  - No developer as gatekeeper
- Target software is the updating script
  - Small python file

The predictions

- Vocational rehabilitation outcome
  - Updated on every patient’s discharge
  - Successful / Unsuccessful
  - Dropout
  - Length
- Next measurement of Icelandic Health-related Quality of Life (IQL)
  - Updated on every submission of questionnaire
  - 12 categories
  - Measured every 3-6 months

Predicting the outcome

- Implemented in June 2016
  - Forgotten about for 10 months
- 72 updates over the 10 month period
  - Reached maximum accuracy early
- All predictions are for events that had not occurred.
  - Real people
  - Real events

Predicting the IQL

- Simulation
  - Bootstrapped accuracy distribution
- Never under 92% accuracy in any IQL subcategory
- Mean accuracy over 99%
Overview

- Introduction
- Fixing Bugs and other examples
- Noteworthy papers and issues
- Getting involved
- Summary and Q&A

A Systematic Study of Automated Program Repair: Fixing 55 out of 105 Bugs for $8 Each

(2012)

Cited -400 times

Automated Software Transplantation

(2015)

Babel Pilgrim: SBSE Can Grow and Craft Entirely New Functionality into a Real World System

(2014)

DNA sequencing consisting of 8,000+ lines of code.

improved version is up to 3x faster

downloaded 1,000 times.

Ported by IBM to one of their super computers

Bowtie2, a DNA sequence alignment/sequence analysis tool

Using Genetic Improvement, Harman and Langdon were capable of increasing performance 70x.
Deep Parameter Optimisation for Face Detection Using the Viola-Jones Algorithm in OpenCV

Bobby R. Bruce\(^1\), Jonathan M. Aitken\(^2\), and Justyna Petko\(^1\)

---

**Face Detection**

- Face
- Not A Face

---

**Face Detection**

- Face
- Not A Face

---

**Integer Literals extracted**

- line 1: a
- line 1: b
- line 10: a
- line 12: a

---

**Original**:

- 191s, 1.04% inaccuracy
- 99s (48% decrease), 1.8% inaccuracy
- 68s (64% decrease), 5.4% inaccuracy
- 46s (76% decrease), 15.4% inaccuracy

---

**Multi-objective optimisation**

---

**PhD Theses**

- David R. White
- Andrea Accuri
- Bobby R. Brown
- Sæmundur Ó. Haraldsson
- Mahmoud R. Bokhari
- And many more to come...
Relationship to other fields

- Optimization/machine learning - OVERFITTING (or: specialisation?)
  (“Is the cure worse than the disease?” Smith et al. FSE 2015)
- Genetic Programming and Metaheuristics
- the automatic design of algorithms
- Automatic parameter tuning/deep parameter tuning/GI

GI & Benchmarking

1. GP suffered a “midlife crisis”
2. Toy problem e.g. lawnmower
5. Is Software Engineering the best benchmark for GP?
6. Do we have a stable set of benchmarks for GI?
   (for program repair: http://program-repair.org/benchmarks.html)
7. Benchmarking is more complex (noise, hardware, prog lang, …)

Measuring Energy

- computational energy consumption growing importance, particularly at the extremes (i.e., mobile devices and datacentres).
  one line = one unit
  simulate (runtime/system calls/) Tools
  Opacitor, PowerGauge
  read battery indicator
  physically measure (e.g. see Bokhari et al.)

Energy Tradeoff

Figure 3: Archive at Generation 240, Experiment 1. The graph shows the trade-offs made by programs within the archive, between total power consumption and error. For both objectives, lower values are better.

820
Source of Genetic Material

1. the program being improved,
2. a different program written in the same language (Petke: MiniSAT competition),
3. a piece of code generated from scratch (GP),
4. different programming language other than the software to be improved.

Theory

- Hard!

- NFL not really valid for GP, and therefore GI.
  - Why – because many programs share same functionality.

$\Rightarrow$ GI will remain empirical for years to come
BREAKDOWN
papers by application

- repair
- runtime
- parallelisation
- energy consumption
- new functionality
- slimming
- memory consumption
- specialisation

Grant Writing

- A grant about GP
  (0%)

VS

- A grant about GI.
  (100%)

Websites

- [http://geneticimprovementofsoftware.com/](http://geneticimprovementofsoftware.com/)
- [http://www.davidrwhite.co.uk/](http://www.davidrwhite.co.uk/)
- [http://daase.cs.ucl.ac.uk/](http://daase.cs.ucl.ac.uk/)
- [http://crest.cs.ucl.ac.uk/publications/](http://crest.cs.ucl.ac.uk/publications/)
- [https://clairelegoues.com/blog/](https://clairelegoues.com/blog/)

Starting point – POP science, GIN, Survey

Genetic Improvement of Software: a Comprehensive Survey

Janyta Polka, Saemundur O. Haraldsson, Mark Harman, William B. Langdon, David R. White, and John R. Woodward

G15#GECCO’19

A Survey of Genetic Improvement Search Spaces

Janyta Polka
Department of Computer Science
University College London
London, UK
janytap@cs.ucl.ac.uk

Saemundur O. Haraldsson
Computing, Science and Engineering
University of Health Sciences at Stirling
Stirling, UK
saemundur@uhs.ac.uk

Mark Harman
School of Computer Science
University of Adelaide
Adelaide, Australia
mark.harman@adelaide.edu.au

William B. Langdon
School of Computer Science
University College London
London, UK
w.langdon@cs.ucl.ac.uk

David R. White
Department of Computer Science
The University of Sheffield
Sheffield, UK
david.white@sheffield.ac.uk

(2017)
Overview

- Introduction
- Fixing Bugs and other examples
- Noteworthy papers and issues
  - Getting involved
- Summary and Q&A

Get involved with GI in No time - or GIN

Available at https://github.com/gintool/gin

v2.0 published in June 2020
“Gin: Genetic Improvement Research Made Easy” (GECCO 2020)

http://www.davidrwhite.co.uk/

Genetic Improvement

- Many success stories
- …however, these typically need at GI expert in the loop
- What’s needed is a more systematic approach
- A toolkit to enable experimentation

“Stupidly simple”

GIN

ECJ

https://cs.gmu.edu/~eclab/projects/ecj/
Gin’s Goals

- Remove incidental difficulties of GI for research and teaching
- Enable focus on general questions
- Provide a central tool for the community
- Support more than bug-fixing: non-functional properties
- Work on open-source software projects out-of-the-box

Vanilla GIN

Version 1.0
gradle/maven support, various types of edits, various samplers, ...

Version 2.0:
gradle/maven support, various types of edits, various samplers, ...

Vanilla GIN: Neighbourhood search

Source-code → AST → AST → Optimised Source

JUnit Test Cases → Apply Patch → Run → Better than before?

Vanilla GIN: Version 2.0:
gradle/maven support, various types of edits, profiler to find “hot” methods, various samplers, ...

The inaugural paper
official V2.0 released on 12 June 2019:
https://github.com/gintool/gin/releases

Gin: Genetic Improvement Research Made Easy

Alexander E.J. Bestwick
Computing Science and Mathematics
University of Strathclyde
mjt@cis.strath.ac.uk

Enda T. Bates
Department of Computer Science
University College London
e.bates@ucl.ac.uk

David R. White
Department of Computer Science
The University of Sheffield
d.r.white@sheffield.ac.uk

ABSTRACT

Gin: Genetic Improvement (GI) is a young field of software engineering, with potential value in improving existing software. Its aim is to promote evolutionary, rather than revolutionary, improvements. This paper describes Gin, a genetic improvement tool for application development. Gin is designed to be simple to use, yet powerful. It supports a variety of features, including

1 INTRODUCTION

Gin: Genetic Improvement (GI) is a young field of software engineering, with potential value in improving existing software. Its aim is to promote evolutionary, rather than revolutionary, improvements. This paper describes Gin, a genetic improvement tool for application development. Gin is designed to be simple to use, yet powerful. It supports a variety of features, including
Gin Design

Gin Pipelines

Preprocessing
- Project Source → EvoSuite → Generated Test Suite → Test Suite Profiling → Target Methods
- Project Test Suite → Test Suite Profiling → Method Tests

Search Space Analysis
- Target Method → Patch Sampling → Dynamic Compilation → JUnit → Patch Profile

Gin Core Classes
Edits

- Edits are single changes to source code
- Building blocks of a repair
- Combined into Patches
- Question: actually, what scale might an edit be?
- Gin supports edits at:
  - line level (Langdon) - delete/replace/copy/swap/move
  - statement level (GenProg) - delete/replace/copy/swap/move
  - constrained (matched) statement - replace/swap
  - micro edits
    - binary & unary operator replacement (OR ↔ AND) (+ + ↔ - -)
    - reorder Boolean expressions (X & Y ↔ Y & X)

Example edits

```java
public class ReplaceStatement extends StatementEdit {

    public int sourceID;
    public int destinationID;

    public ReplaceStatement(SourceFileTree sf, Random r) {
        sourceID = sf.getRandomStatementID(false, r);
        destinationID = sf.getRandomStatementID(true, r);
    }

    public SourceFile apply(SourceFileTree sf) {
        Statement source = sf.getStatement(sourceID);
        Statement dest = sf.getStatement(destinationID);
        return sf.replaceNode(dest, source.clone());
    }
}
```

Example edits

```java
public class MatchedReplaceStatement extends ReplaceStatement {

    public MatchedReplaceStatement(SourceFileTree sf, Random r) {
        super(0, 0);
        destinationID = sf.getRandomStatementID(true, r);
        sourceID = sf.getRandomNodeID(false);
    }

    public void getStatement(destinationID, getClass(), r);
}
```
Gin invokes test cases via JUnit…

Tracks:
- compile success;
- run-time errors, exception types
- actual & expected outcomes
- timing: wall-clock and CPU time

Note: If you prefer to use the more "traditional" way of writing the file to disk first - e.g., due to integration of Gin into other pipelines - then you can use a command-line flag to do so.

Sampling

- Included samplers:
  - EmptyPatchTester
  - RandomSampler
  - DeleteEnumerator
  - LocalSearch

- Possible Questions:
  - What is the effectiveness of a given edit type for fixing a category of bug?
  - How robust is the space of single-line edits, modulo the given test suite?
  - ...

The following is one really wide output file:
Local search

```
package Triangle;

public class Triangle {
    public static int classifyTriangle(int a, int b, int c) {
        int[] sides = new int[]{a, b, c};
        Arrays.sort(sides);
        if (sides[0] + sides[1] <= sides[2]) {
            return 0; // Invalid
        }
            return 2; // Equilateral
        }
            return 3; // Isosceles
        }
        return 1; // Scalene
    }
}
```
Maven and Gradle API documentation is sparse!
- And many projects seem to break conventions about paths, resources etc.
- **Project** class wraps most of what we have learned
  - provide the classpath for a project
  - find a particular source file within a project’s file hierarchy
  - provide a standard method signature for a given method
  - provide a list of project tests
  - run a unit test given its name
- Gin can infer the necessary classpath and dependencies for running unit tests from a Maven or Gradle project, or these can be specified manually
- Maven projects can be updated automatically with new unit tests from **EvoSuite**

### Examples with jCodec (maven project)

- **Profiler**
  ```
  $projectnameforgin='jcodec';
  java -Dtinylog.level=trace -cp ../../ginfork/build/gin.jar gin.util.Profiler
  -h ~/.sdkman/candidates/maven/current/ -p $projectnameforgin -d 
  -o $projectnameforgin.Profiler_output.csv -r 1
  ```

- **EmptyPatchTester**
  ```
  $projectnameforgin='jcodec';
  java -Dtinylog.level=trace -cp ../../ginfork/build/gin.jar gin.util.EmptyPatchTester
  -h ~/.sdkman/candidates/maven/current/ -p $projectnameforgin -d 
  -m $projectnameforgin.Profiler_output.csv
  -o $projectnameforgin.EmptyPatchTester_output.csv
  ```
Examples with jCodec (maven project)

- Profiler
  `projectnameforgin='jcodec';
  java -Dtinylog.level=trace -cp ../../ginfork/build/gin.jar gin.util.Profiler
  -h ~/.sdkman/candidates/maven/current/ -p $projectnameforgin -d .
  -o $projectnameforgin.Profiler_output.csv`

- EmptyPatchTester
  `projectnameforgin='jcodec';
  java -Dtinylog.level=trace -cp ../../ginfork/build/gin.jar gin.util.EmptyPatchTester
  -h ~/.sdkman/candidates/maven/current/ -p $projectnameforgin -d .
  -m $projectnameforgin.Profiler_output.csv
  -o $projectnameforgin.EmptyPatchTester_output.csv`

- PatchSampler
  `projectnameforgin='jcodec';
  java -Dtinylog.level=trace -cp ../../ginfork/build/gin.jar gin.util.PatchSampler
  -h ~/.sdkman/candidates/maven/current/ -p $projectnameforgin -d .
  -m $projectnameforgin.Profiler_output.csv
  -o $projectnameforgin.PatchSampler_LINE_output.csv
  -editType LINE -patchNo 100`

- Generate tests
  `java -cp build/gin.jar gin.util.TestCaseGenerator -projectDir ../casestudies/RxJava
  -projectName RxJava -evosuiteCP libs/evosuite-1.0.6.jar -generateTests -classNumber 3
  -projectTarget ../casestudies/RxJava/build/classes/java/main`

Available at
https://github.com/gintool/gin

The team actively uses Gin to push the GI boundaries, and quite a few papers are in the works.

Open for contributions!
- Particularly new edits and tools
  https://github.com/gintool/gin
- we’d like this to become the MiniSAT of GI

Comments/questions: Sandy (Alexander E.I. Brownlee) sbro@cs.stir.ac.uk

Overview

- Introduction
- Fixing Bugs and other examples
- Noteworthy papers and issues
- Getting involved

Summary and Q&A

Genetic Improvement vs Genetic Programming

1. Start from an existing program
2. BLOAT? – interpretation?
3. NO function / terminal set
4. Improvement of non-functional properties.
5. Easier to write grants
6. Different benchmarks.
7. Population of edits NOT programs.
Let’s start with existing programs. Not like standard GP.

Python vs C vs Java? Amenable to GI? Most popular

Benchmarking ???

Population of edits, not programs

GP applied to real software

Large, loops, side-effect, modules,…

Non functional properties

Questions?

Saæmundur (Saemi) Haraldsson <soh@cs.stir.ac.uk>
John Woodward <j.woodward@gmul.ac.uk>
Markus Wagner <markus.wagner@adelaide.edu.au>