Flight of the FINCH through the Java Wilderness

Michael Orlov and Moshe Sipper
orlovm, sipper@cs.bgu.ac.il

Department of Computer Science
Ben-Gurion University, Israel

2010 HUMIES
July 9, GECCO, Portland
“While it is common to describe GP as evolving programs, GP is not typically used to evolve programs in the familiar Turing-complete languages humans normally use for software development. It is instead more common to evolve programs (or expressions or formulae) in a more constrained and often domain-specific language.”

A Field Guide to Genetic Programming
[Poli, Langdon, and McPhee, 2008]
Our Goals

From programs...

Evolve actual programs
written in Java

...to software!

Improve (existing) software
written in unrestricted Java
FINCH: Fertile Darwinian Bytecode Harvester

- A system for evolving Java (bytecode).
- Employs a sophisticated compatible crossover operator.
- **Always** produces correct (compilable) bytecode.
- Which we can deploy directly or decompile back to Java for perusal.
- All you need: A seed and a wish (= fitness function).
- If the wish is good—even a **bad** seed will eventually blossom!
“On two occasions I have been asked, ‘Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?’ I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.”

Charles Babbage

Garbage in—Garbage out...

Or maybe not?
“On two occasions I have been asked, ‘Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?’ I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.”

Charles Babbage

Garbage in—Garbage out...
“On two occasions I have been asked, ‘Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?’ I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.”

Charles Babbage

Garbage in—Garbage out...
Sample seed supplied to FINCH:

```java
class SimpleSymbolicRegression {
    Number simpleRegression(Number num) {
        double x = num.doubleValue();
        double llsq = Math.log(Math.log(x*x));
        double dv = x / (x - Math.sin(x));
        double worst = Math.exp(dv - llsq);
        return Double.valueOf(worst + Math.cos(1));
    }
}

/* Rest of class omitted */
```

Evolved program computing $x^4 + x^3 + x^2 + x$:

class SimpleSymbolicRegression_0_7199 {
    Number simpleRegression(Number num) {
        double d = num.doubleValue();
        d = num.doubleValue();
        double d1 = d; d = Double.valueOf(d + d * d * num.doubleValue()).doubleValue();
        return Double.valueOf(d +
            (d = num.doubleValue()) * num.doubleValue());
    }

    /* Rest of class unchanged */
}
Evolved program computing $x^9 + x^8 + \cdots + x^2 + x$:

```java
Number simpleRegression(Number num) {
    double d = num.doubleValue();
    return Double.valueOf(d + (d * (d * (d +
       ((d = num.doubleValue()) +
       (((num.doubleValue() * (d = d) + d) *
         d + d) * d + d) * d) * d) + d) + d) * d);
}
```
Evolved program solving Artificial Ant problem:

```java
void step() {
    if (foodAhead()) {
        move(); right();
    }
    else {
        right(); right();
        if (foodAhead())
            left();
        else {
            right(); move();
            left();
        }
    }
    left(); left();
}
```
Evolved program solving Intertwined Spirals problem:

\[
\text{uses sign of } \sin\left(\frac{9}{4}\pi^2 \sqrt{x^2 + y^2} - \tan^{-1} \frac{y}{x}\right)
\]

```java
boolean isFirst(double x, double y) {
    double a, b, c, e;
    a = Math.hypot(x, y);  e = y;
    c = Math.atan2(y, b = x) +
        -(b = Math.atan2(a, -a))
        * (c = a + a) * (b + (c = b));
    e = -b * Math.sin(c);
    if (e < -0.0056126487018762772) {
        b = Math.atan2(a, -a);
        b = Math.atan2(a * c + b, x);  b = x;
        return false;
    }
    else
        return true;
}
```
Compare with Koza’s best-of-run:

\[
\sin(\text{iflte}(\text{iflte}(+Y Y)(+X Y)(-X Y)(+Y Y))(*X X))
\sin(\text{iflte}(%Y Y)(%\sin(\sin(%Y 0.30400002)))X)(%Y 0.30400002)(\text{iflte}(\text{iflte}(\sin(%(%Y (+X Y)))
0.30400002))(+X Y))(%X -0.1039997)(-X Y)(*(+ -0.12499994 -0.15999977)(-X Y))0.30400002(\sin(\sin(\text{iflte}(\sin(%(%Y 0.30400002)0.30400002))(+X Y))
(%(\sin Y)Y)(\sin(\sin(%(\sin X)(+ -0.12499994
-0.15999977)))))(%(+(+X Y)(+Y Y))0.30400002)))
(+(+X Y)(+Y Y))))(\sin(\text{iflte}(\text{iflte} Y (+X Y)(-X Y)
(+Y Y))(*X X)(\sin(\text{iflte}(%Y Y)(%\sin(\sin(%Y
0.30400002)))X)(%Y 0.30400002)(\sin(\sin(\text{iflte}(\text{iflte}
\sin(%(\sin X)(+ -0.12499994 -0.15999977))))(%X
-0.1039997)(-X Y)(+X Y))(\sin(%(\sin X)(+
-0.12499994 -0.15999977)))(\sin(\sin(%(\sin X)(+
-0.12499994 -0.15999977))))(+(+X Y)(+Y Y)))))))(%Y
0.30400002))))
\]
Tree GP vs. FINCH

And compare the phenotypes:

Koza’s:

Ours:
Evolved program computing sum of values in array: (loop solution)

```java
int sumlist(int list[]) {
    int sum  = 0;
    int size = list.length;
    for (int tmp = 0; tmp < list.length; tmp++) {
        size = tmp;
        sum = sum - (0 - list[tmp]);
    }
    return sum;
}
```

Yes, FINCH can handle loops...
Evolved program computing sum of values in array: (List solution)

```java
int sumlist(List list) {
    int sum = 0;
    int size = list.size();
    for (Iterator iterator = list.iterator();
         iterator.hasNext(); ) {
        int tmp = ((Integer) iterator.next()).intValue();
        tmp = tmp + sum;
        if (tmp == list.size() + sum)
            sum = tmp;
    }
    return sum;
}
```

Yes, FINCH can handle Java 5.0 constructs...
Evolved program computing sum of values in array: (recursive solution)

```java
int sumlistrec(List list) {
    int sum = 0;
    if (list.isEmpty())
        sum = sum;
    else
        sum += ((Integer)list.get(0)).intValue() +
            sumlistrec(list.subList(1, list.size()));
    return sum;
}
```

Yes, FINCH can handle recursion...
Enough Garbage In!

- We can turn **bad** (seeds) into **good** (programs)

- **Input:** Good program implementing sophisticated Minimax algorithm to play Tic-Tac-Toe.
- **Problem:** Programmer made a small, insidious, very hard-to-detect error.
- Can FINCH save the day?

- We implemented four errors: All were easily swept away by FINCH.
Enough Garbage In!

- We can turn bad (seeds) into good (programs)

- Input: **Good** program implementing sophisticated Minimax algorithm to play Tic-Tac-Toe.
- Problem: Programmer made a small, insidious, very hard-to-detect error.
- Can FINCH save the day?

- We implemented four errors:
  All were easily swept away by FINCH.
Enough Garbage In!

• We can turn **bad** (seeds) into **good** (programs)

• Input: **Good** program implementing sophisticated Minimax algorithm to play Tic-Tac-Toe.

• Problem: Programmer made a small, insidious, very hard-to-detect error.

• Can FINCH save the day?

• We implemented four errors: All were easily swept away by FINCH.
```java
1 int negamaxAB(TicTacToeBoard board,
2     int alpha, int beta, boolean save) {
3     Position[] free = getFreeCells(board);
4     // utility is derived from the number of free cells left
5     if (board.getWinner() != null)
6         alpha = utility(board, free);
7     else if (free.length == 0)
8         alpha = 0, save = false;
9     else for (Position move: free) {
10        TicTacToeBoard copy = board.clone();
11        copy.play(move.row(), move.col(),
12                    copy.getTurn());
13        int utility = -negamaxAB(copy,
14                                  -beta, -alpha, false, save);
15        if (utility > alpha) {
16           alpha = utility;
17           if (save)
18               // save the move into a class instance field
19               chosenMove = move;
20           if (alpha >= beta, beta >= alpha)
21               break;
22        }
23    }
24    return alpha;
25 }
```
Result is Human-Competitive

(G) The result solves a problem of indisputable difficulty in its field.

- There is a widely recognized need for automatic improvement of existing software.
- Improving software is indisputably difficult (G) (Did anybody say ‘difficult’? Merely ‘difficult’?)
- No technique previously existed that allowed the automatic improvement of unrestricted software written in a widely used, real-life programming language.
- And along came FINCH...
(D) The result is publishable in its own right as a new scientific result independent of the fact that the result was mechanically created.

- The evolved programs are fully functional Java programs solving hard problems.
- The FINCH system itself is of indisputable interest to the software industry at large.
Why is Result Best?

• Our work aims at one of the hardest problems known to (and created by) man: software design.

• Given the size and importance of the software industry, any step taken toward automating the programmer’s task could impact society in ways far outreaching the boundaries of evolutionary computation.
Why is Result Best? (cont’d)

- Very little work within the field of search-based software engineering tackles the issue of evolving **actual** software written in **mainstream** languages.

- The work of Weimer and Forrest on automatically evolving patches to fix buggy C programs is probably the most worthy of mention.

- However, their system is currently limited to evolving small patches in predetermined source locations.

- Works when fixing bugs with known positive and negative test cases, which afford the ability to focus the search using standard software engineering techniques.
• We have taken another major step forward.
• Ours is the first approach that allows viable evolution of extant, real-world software in a mainstream programming language (Java is one of the 2 most popular programming languages).
• Moreover, FINCH is not limited to Java: Scala, Groovy, Jython, Kawa, JavaFX Script, Clojure.
Why is Result Best? (cont’d)

• “Judiciously used, digital evolution can substantially augment the cognitive limits of human designers and can find novel (possibly counterintuitive) solutions to complex ... system design problems.”

   (Recent study by US DoD on futuristic systems)

• FINCH represents a significant step on the (long) path toward full-fledged Darwinian Software Development.
Why is Result Best? (cont’d)

• “Judiciously used, digital evolution can substantially augment the cognitive limits of human designers and can find novel (possibly counterintuitive) solutions to complex ... system design problems.”

   (Recent study by US DoD on futuristic systems)

• FINCH represents a significant step on the (long) path toward full-fledged Darwinian Software Development.
“I believe that in about fifty years’ time it will be possible, to programme computers ... to make them play the imitation game so well that an average interrogator will not have more than 70 per cent. chance of making the right identification after five minutes of questioning.”

A. M. Turing, “Computing machinery and intelligence,”
Mind, 59(236), 433-460, Oct. 1950
“... despite its current widespread use, there was, within living memory, equal skepticism about whether compiled code could be trusted. If a similar change of attitude to evolved code occurs over time ...”

M. Harman, “Automated patching techniques: The fix is in,”
We believe that in about fifty years’ time it will be possible, to program computers... by means of evolution.

Not merely possible but indeed prevalent.

Turing was wrong—will we be?
We believe that in about fifty years’ time it will be possible, to program computers... by means of evolution.

Not merely possible but indeed prevalent.

Turing was wrong—will we be?
We believe that in about fifty years’ time it will be possible, to program computers... by means of evolution.

Not merely **possible** but indeed **prevalent**.

Turing was wrong—will we be?
2060 (cont’d)

To find out, please register for GECCO 2060.
References
