Teach Your Computer to Code FizzBuzz

Frances Buontempo

@fbuontempo
frances.buontempo@gmail.com
overload@accu.org
https://pragprog.com/book/fbmach/genetic-algorithms-and-machine-learning-for-programmers

Chris Simons

@chrislsimons
chris.simons@uwe.ac.uk
www.fet.uwe.ac.uk/~clsimons/

UWE Bristol | University of the West of England

Workshop materials can be found at

http://www.fet.uwe.ac.uk/~clsimons/ACCU2019/

JCLEC requires Java SE Development Kit, e.g. version 8

https://www.oracle.com/technetwork/java/javase/overview/index.html

Workshop

Evolutionary computing

Frameworks for evolutionary computing

Java Class Library for Evolutionary Computing (JCLEC)

Optimisation problems:

1 - ‘OneMax’ Problem

2 - How to program your way out of a paper bag

3 - FizzBuzz

Genetic Programming, Genetic Improvement

Slides

Evolutionary computing

Frameworks for evolutionary computing

Coding

Java Class Library for Evolutionary Computing (JCLEC)

Optimisation problems:

1 - ‘OneMax’ Problem

2 - How to program your way out of a paper bag

3 - FizzBuzz

Slides

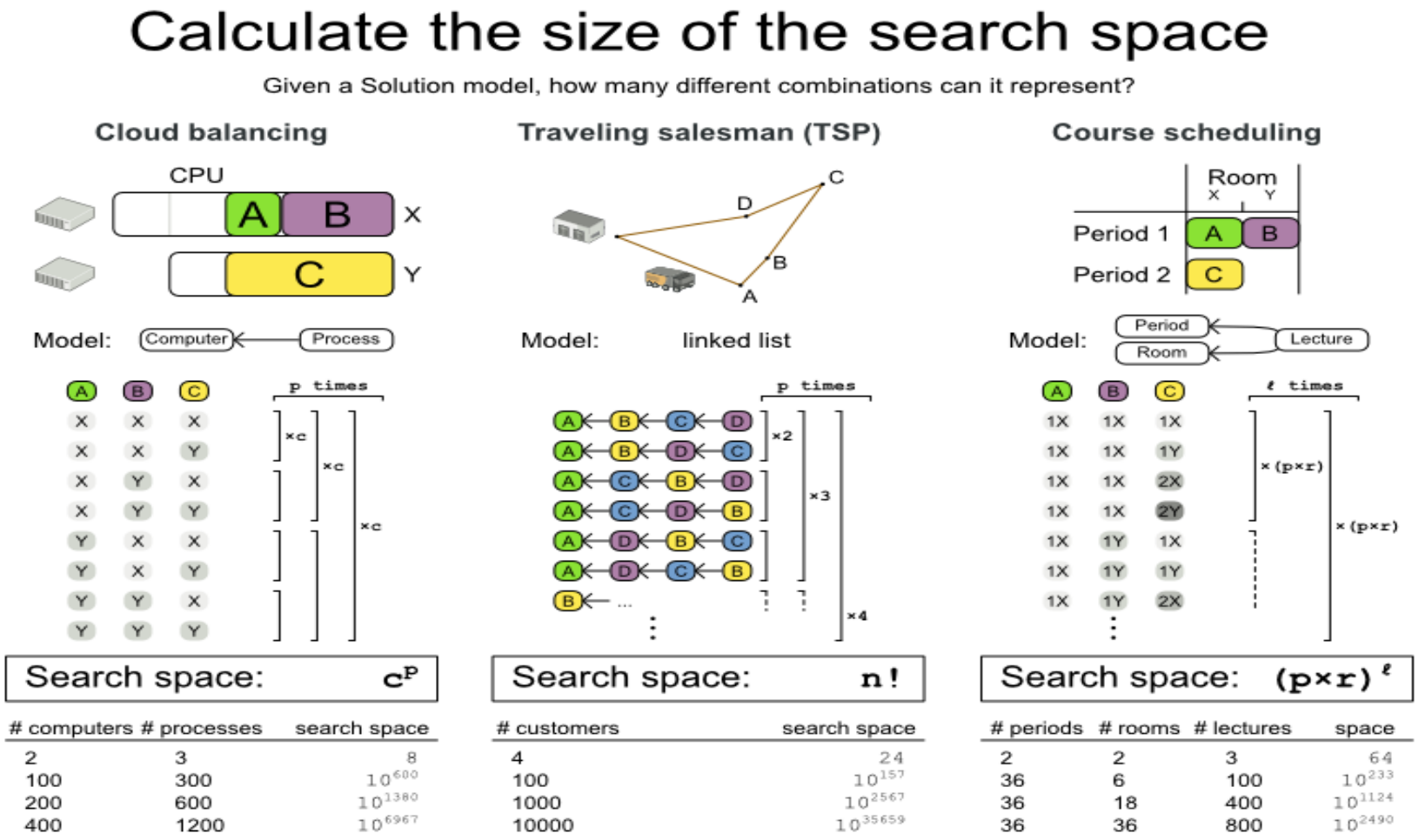
Genetic Programming, Genetic Improvement

When we want to optimise:

- We might want to maximise ‘quality’, and/or
- We might want to minimise ‘cost’…

We want to find the ‘best’ (for whatever ‘best’ means for our situation)

So what's the problem? *combinatorial explosion*



http://www.optaplanner.org/blog/2014/03/27/searchSpaceSizeCalculation.png

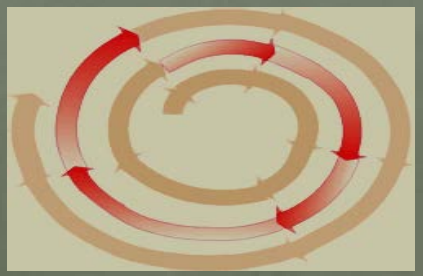
What can we do about it?

natural evolution

i.e. the change in the inherited characteristics of biological populations over successive generations.

environment

Selection of fittest individuals



sexual reproduction for population diversity / variety

www.flickr.com/photos/armincifuentes/

Computational evolution

Representation of an “individual” solution

e.g. arrays, trees, models, code etc. etc.

```
initialise population at random
evaluate each individual
while( not done )
  select parents
  recombine pairs of parents
  mutate new candidate individuals
  evaluate each individual
  select candidates for next generation
end while
```

Ideas from biology (1)

Information concerning the characteristics of a solution ***individual*** is encoded in ‘genes’ – all the gene values of an individual is known as the ***genotype***.

Typically, many individuals make up a ***population***.

Individuals can become ***parents*** from whom ***offspring*** are created. The offspring help to form the new ***generation***, and can themselves become parents in the next generation. Evolutionary algorithms can run for many generations, until some ***termination condition***.

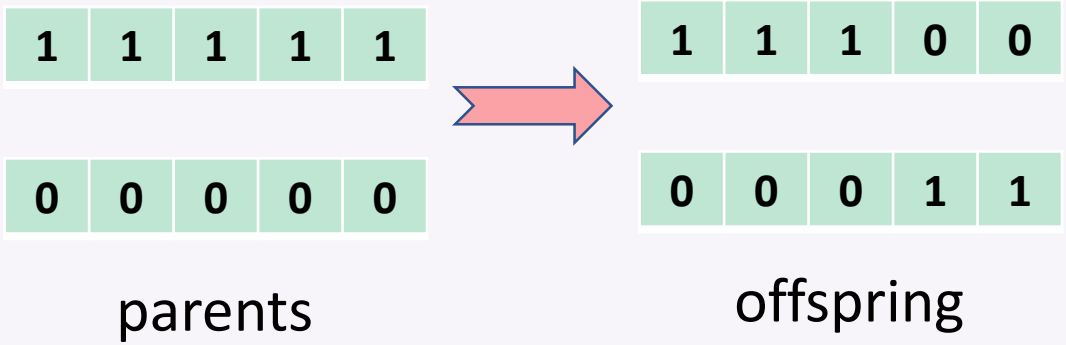
Ideas from biology (2)

Evaluation of a solution ***individual*** gives some ***fitness*** value or ***cost*** value that is to be optimised, either ***maximised*** or ***minimised***.

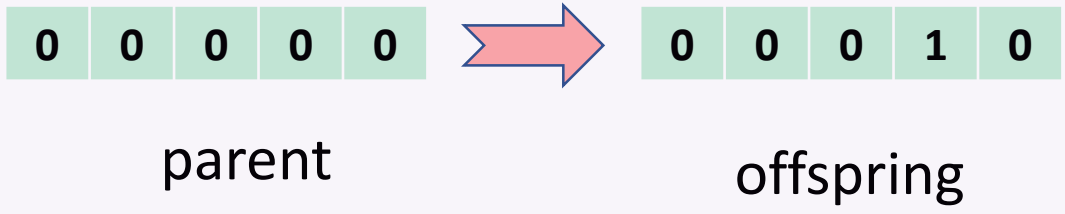
Only the fittest solution ***individuals*** are ***selected*** to breed ***offspring;*** individuals can enter a ***tournament***, the fittest wins the right to breed.

Diversity in the ***population*** is maintained by:

Recombination (sexual reproduction)



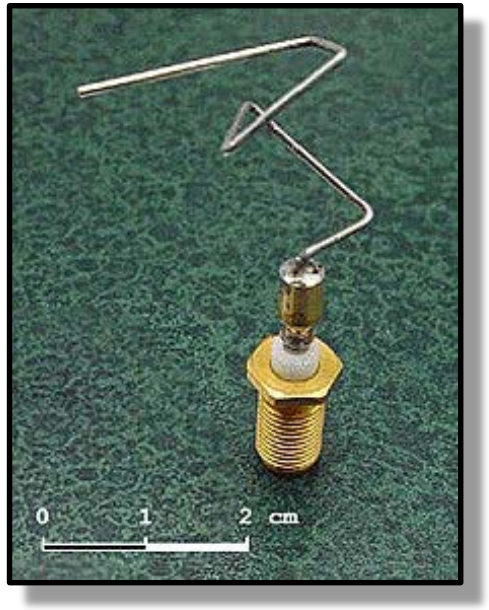
Mutation (asexual reproduction)



Many applications of Evolutionary Computing

Examples include many well-known optimisation problems such as

- course timetabling,
- nurse rostering,
- process scheduling,
- network routing,
- vehicle delivery scheduling,
- load balancing,
- Etc. etc.



The 2006 NASA ST5 spacecraft antenna. This complicated shape was found by an evolutionary computer design program to create the best radiation pattern.

https://en.wikipedia.org/wiki/Evolved_antenna

Workshop

Evolutionary computing

Frameworks for evolutionary computing

Java Class Library for Evolutionary Computing (JCLEC)

Optimisation problems:

1 - ‘OneMax’ Problem

2 - How to program your way out of a paper bag

3 - FizzBuzz

Genetic Programming, Genetic Improvement

Frameworks for Evolutionary Computing

Characteristics include:

- mechanisms for the integration of problem-specific knowledge, such as problem constraints and fitness function(s);
- components to configure and monitor the execution, allowing the user to set execution parameters and visualise intermediate results;
- designed with ***best practices*** and ***design patterns*** in mind.

Language	Framework	version	Date
C++	Evolutionary Computation Framework (ECF)	1.4.2	2017
	Evolving Objects (EO)	1.3.1	2012
	jMetalCpp	1.7	2016
	Mallba	2.0	2009
	Open Beagle	3.0.3	2007
	OptFrame	2.2	2017
	PaGMO	2.5	2017
	ParadisEO	2.0.1	2012
Java	Java-based Evolutionary Computation Research System (ECJ)	24.0, 25.0	2017
	Evolutionary Algorithms Workbench (EvA)	2.2.0	2015
	Java Class Library for Evolutionary Computation (JCLEC)	4.0	2014
	jMetal	5.3	2017
	Multi-Objective Evolutionary Algorithm (MOEA) Framework	2.12	2017
	Opt4J	3.1.4	2015
C#	GeneticSharp (A C# Genetic Algorithm Library)	On-going	2017
	HeuristicLab (A Paradigm-Independent and Extensible Environment for Heuristic Optimization)	3.3.14	2016
Python	Distributed Evolutionary Algorithms in Python (DEAP)	1.1.0	2017
	jMetalPy	On-going	2017
	Pyevolve	0.6rc_1	2015
	PyGMO	On-going	2017
	Pyvolution	1.1	2012
Matlab	Genetic and Evolutionary Algorithm Toolbox for Matlab (GEATbx)	3.8	2017
	Global Optimisation Toolbox	R2017b	2017
	Matlab Platform for Evolutionary Multi-objective Optimisation (PlatEMO)	1.3	2017

Many frameworks available!

For further details, see Overload 142

https://accu.org/index.php/journals/c380/

Workshop

Evolutionary computing

Frameworks for evolutionary computing

Java Class Library for Evolutionary Computing (JCLEC)

Optimisation problems:

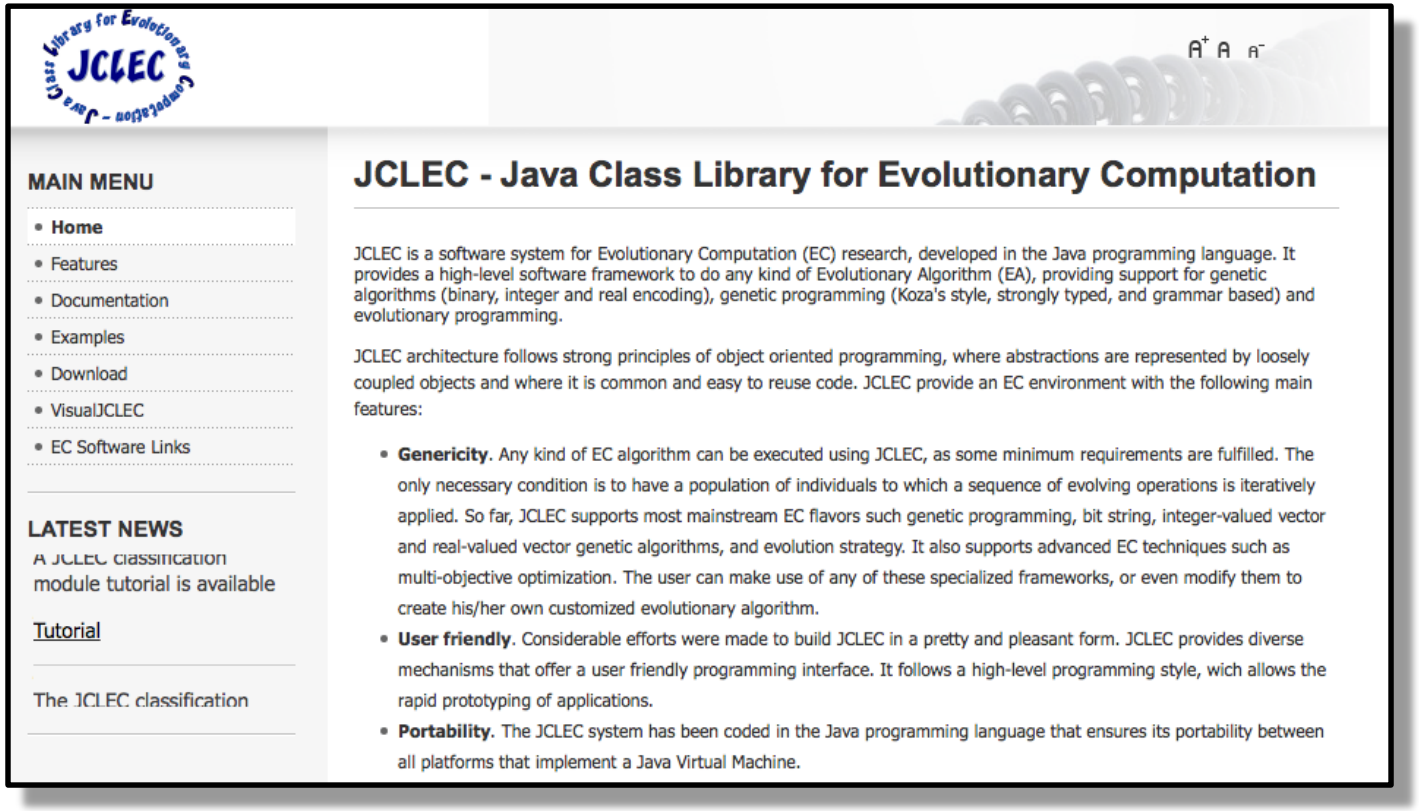
1 - ‘OneMax’ Problem

2 - How to program your way out of a paper bag

3 - FizzBuzz

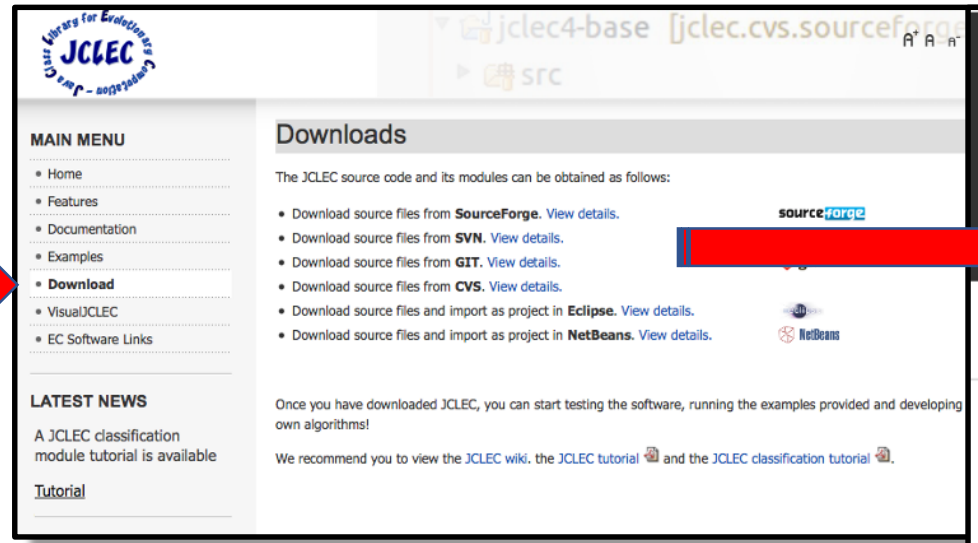
Genetic Programming, Genetic Improvement

Time to look at an example of a evolutionary computing framework



http://jclec.sourceforge.net

Time to download the framework http://jclec.sourceforge.net



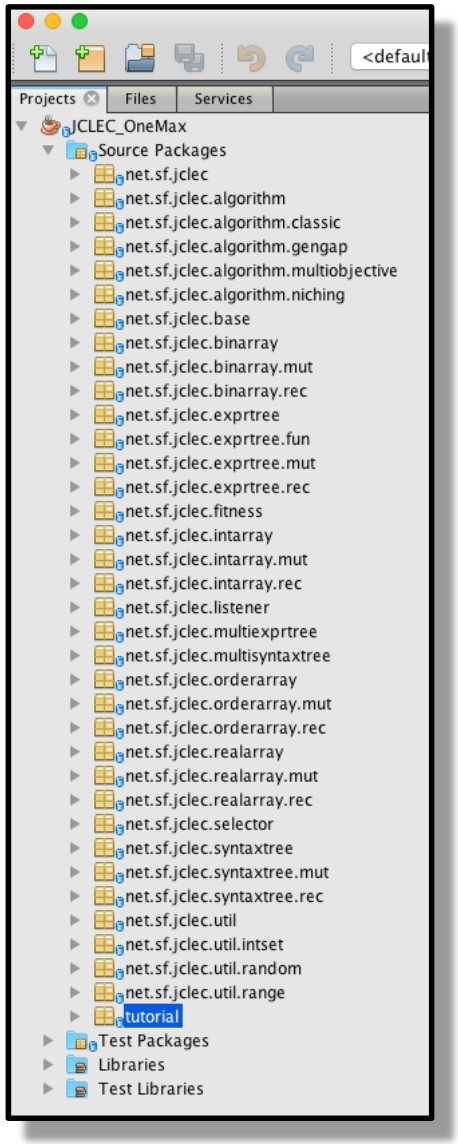
We need this **FIRST**

But we don't need this

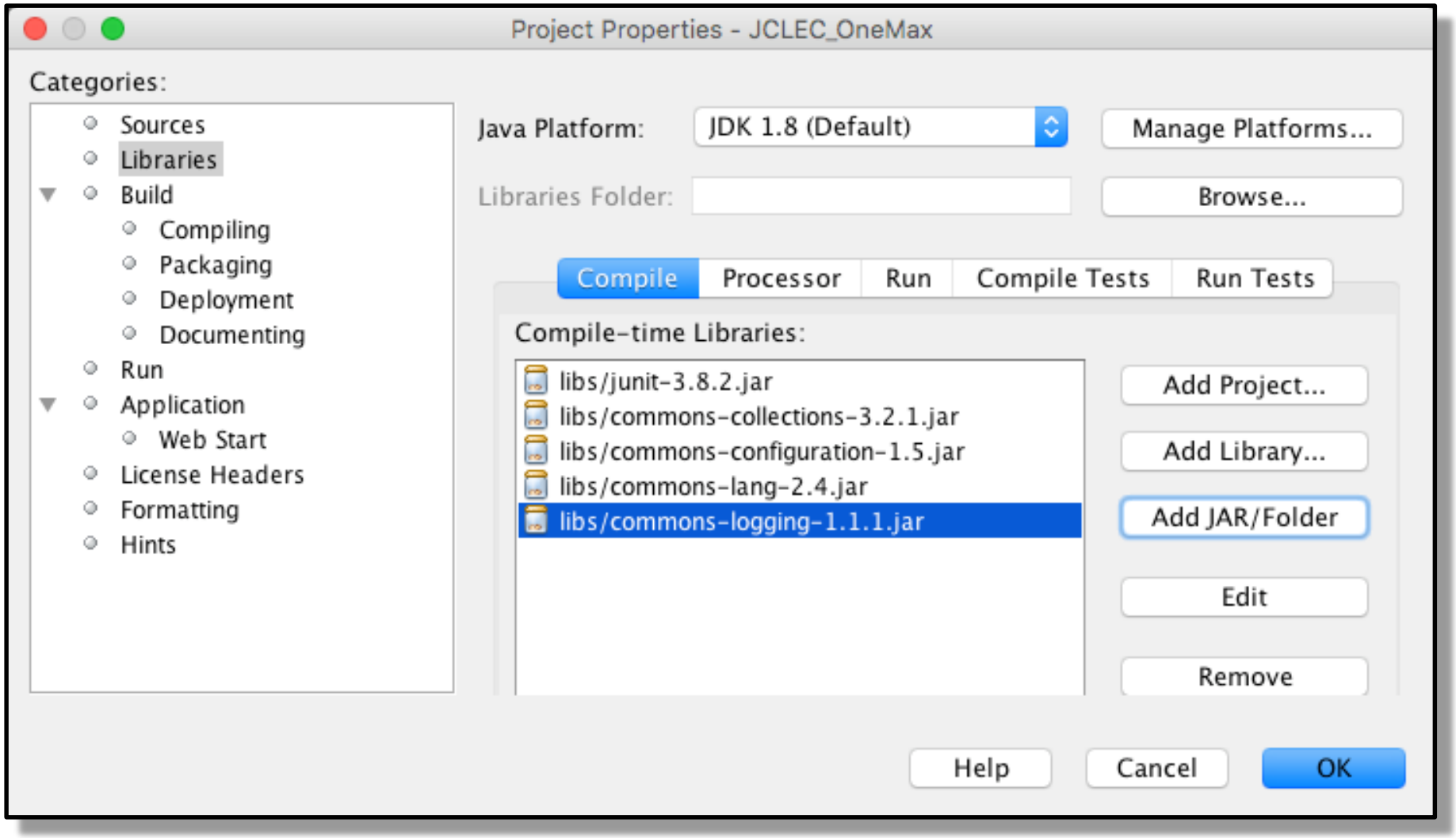
And we need this **SECOND**

You can copy an Eclipse or NetBeans project, or

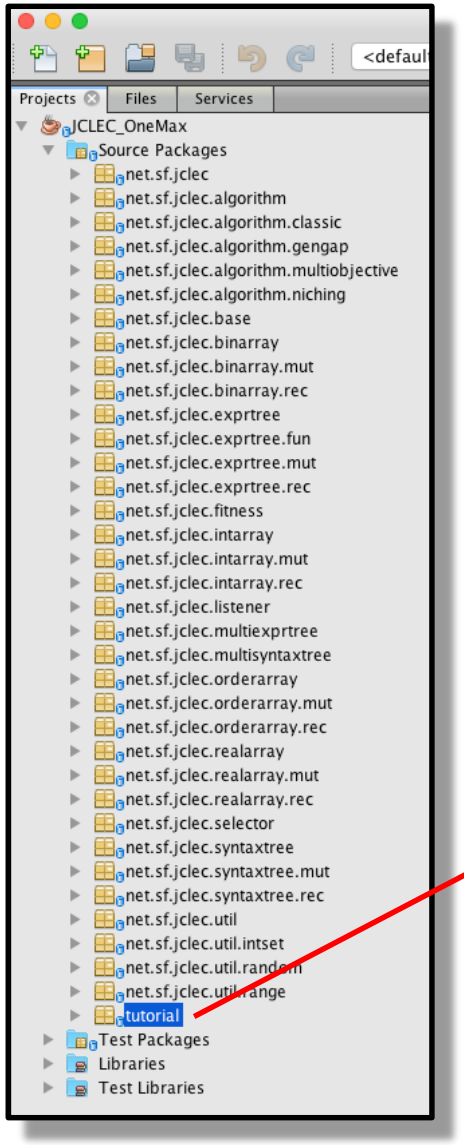
From **jclec4_base.zip**, copy extracted JCLEC source files to an IDE of your choosing, e.g.



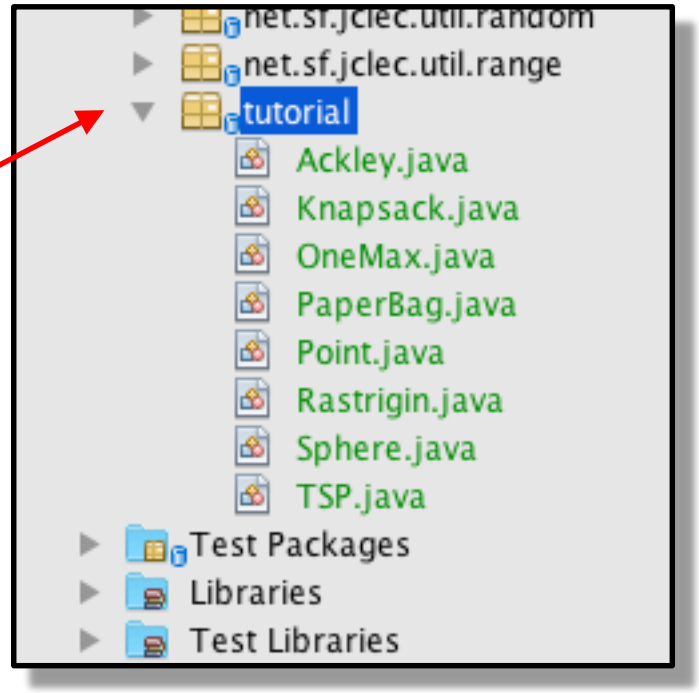
From **jclec4_base.zip**, extract and let the IDE know about the required libraries, e.g.



From jclec4-tutorial.zip,



place tutorial source files
in a package called ‘tutorial’



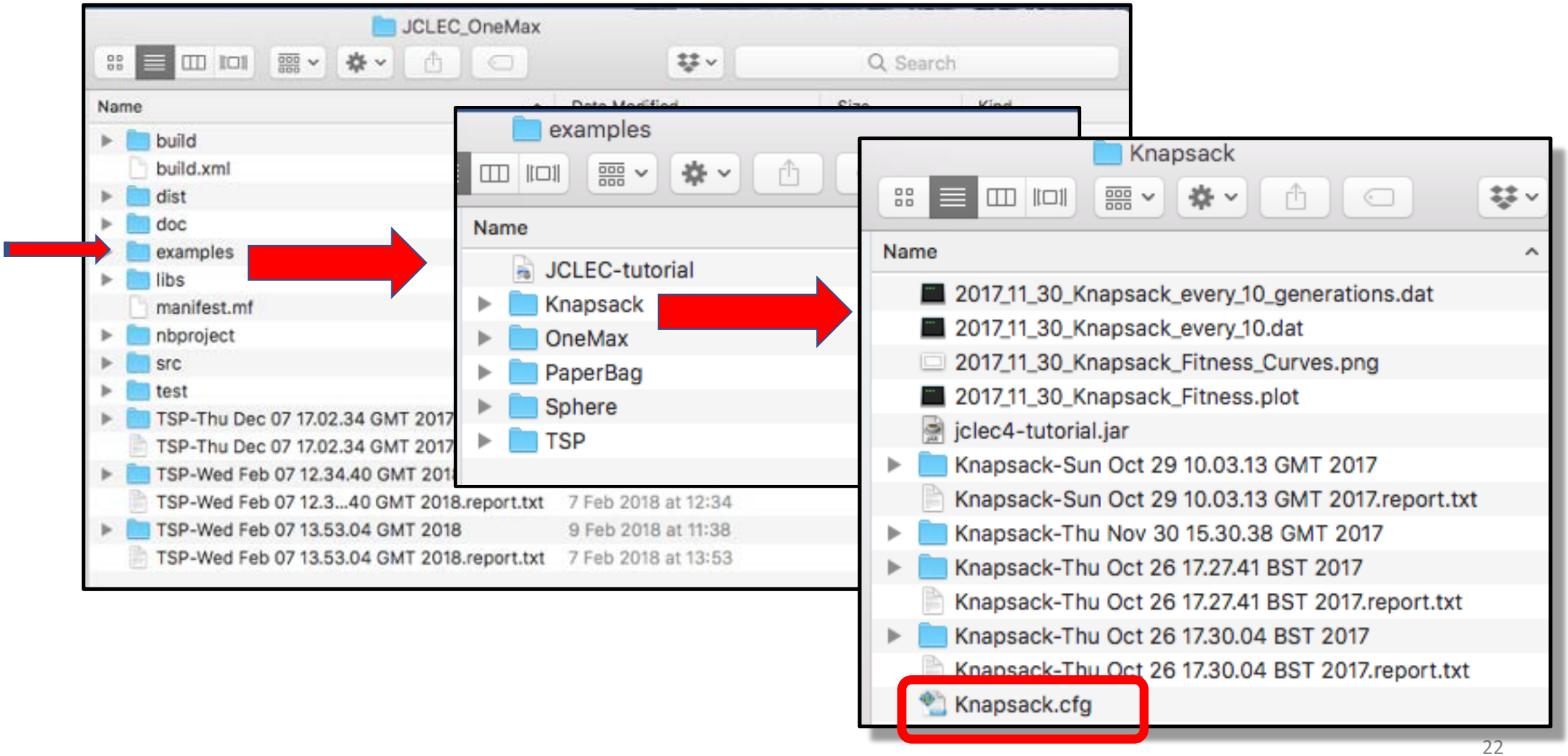
Each JCLEC project has an XML-based configuration file to specify:

1. Evolutionary algorithm components used, and

2. parameter set up

XML configuration files have a `.cfg` suffix

Also from **jclec4-tutorial.zip**, copy tutorial example configuration files to a folder called ‘examples’, e.g.



Note

- For the Travelling Salesman Problem (TSP) example, you may also need `orderarray` package from **jclec4-tutorial.zip**

Workshop

Evolutionary computing

Frameworks for evolutionary computing

Java Class Library for Evolutionary Computing (JCLEC)

Optimisation problems:

1 - ‘OneMax’ Problem

2 - How to program your way out of a paper bag

3 - FizzBuzz

Genetic Programming, Genetic Improvement

The ”hello world” of evolutionary computing!

The OneMax problem consists of maximising the number of ones in a bitstring.

Let’s take a length of 100 bits for the bitstring.

e.g. http://tracer.lcc.uma.es/problems/onemax/onemax.html

Yes, I know, we can do this in our heads :) but it’s a good example of getting going with the framework…

Algorithm design and parameter set up – let's apply some patterns…

Representation

- how to encode a candidate solution?

a binary array

Fitness

- how to evaluate the fitness of a candidate solution?

count the number of 1s

Diversity

- how to make offspring different to parents?

crossover and mutation

Initialisation: random

Evolution: simple generational with elitism (SGE)

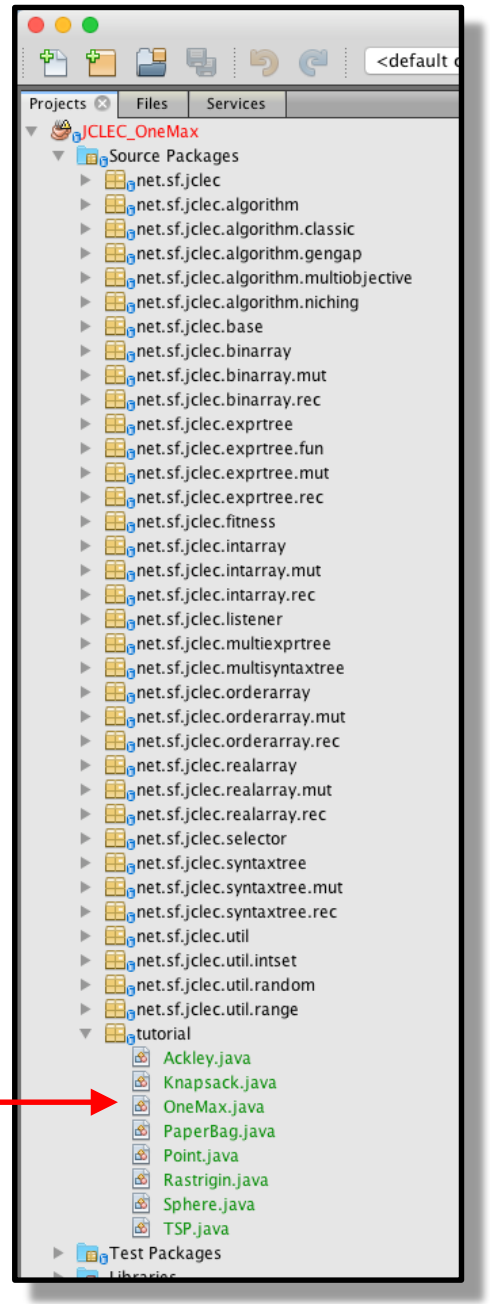
… and suggested parameters

Population size: 100 individuals

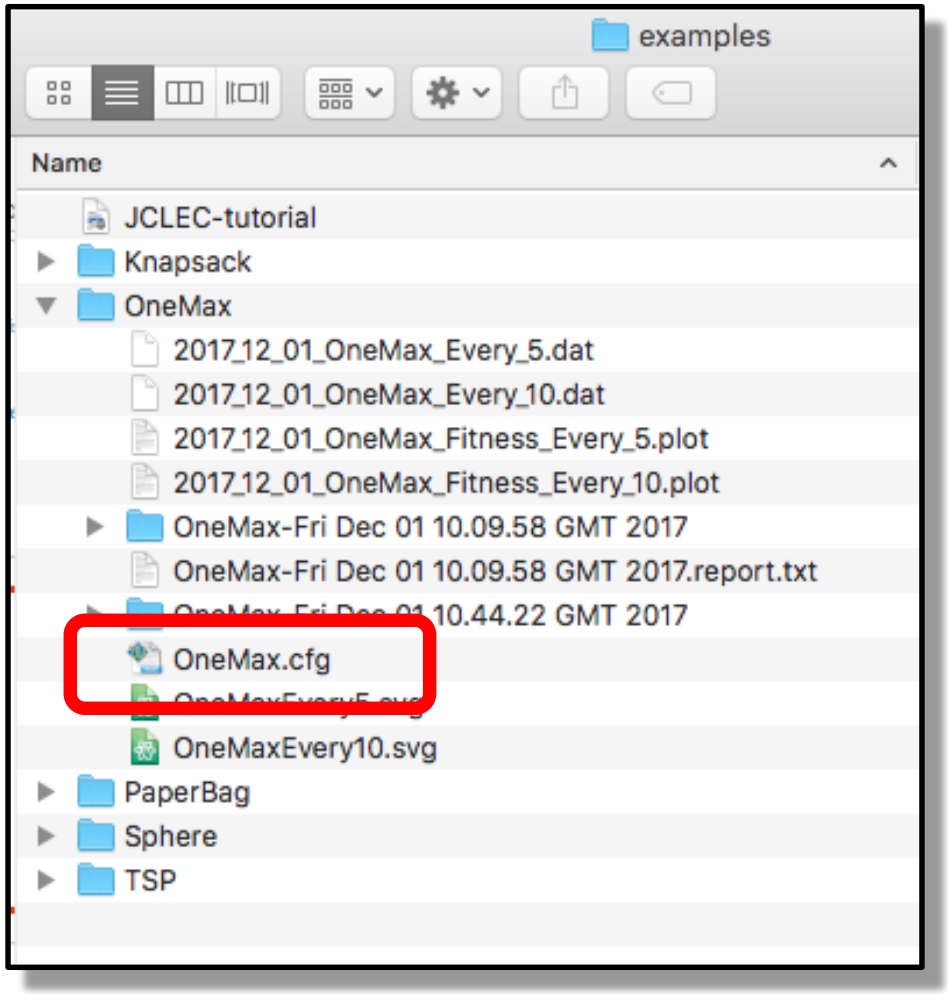
Stop Criterion: 50 generations

Parent selection: tournament of 2 individuals

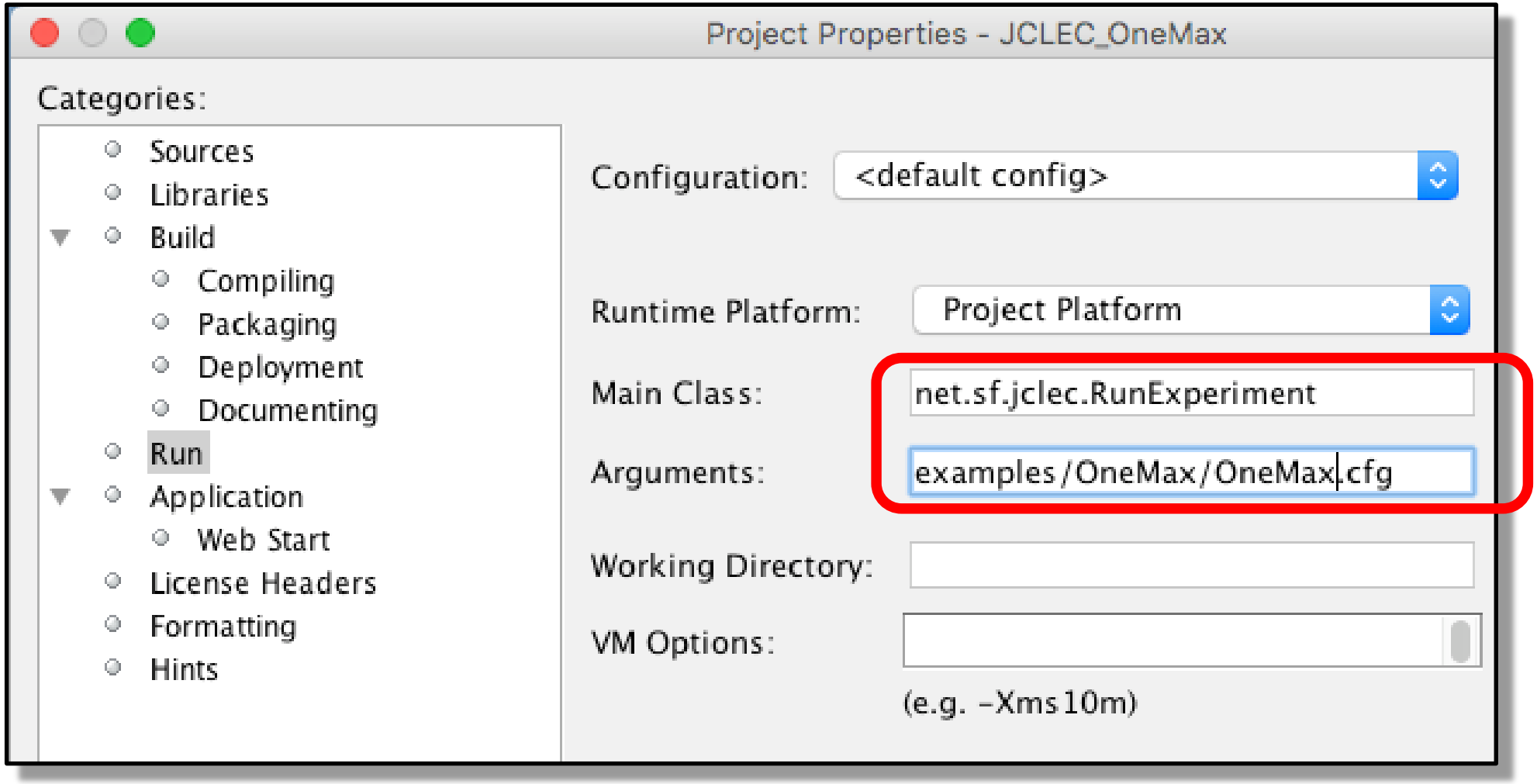
Make a new file
“OneMax.java”



And make a new folder “OneMax” in ”examples” for the configuration file – “OneMax.cfg”



Don’t forget to invoke the executable with ”OneMax.cfg” as an argument



Enjoy the programming!

Here’s one way to solve the OneMax optimisation problem…

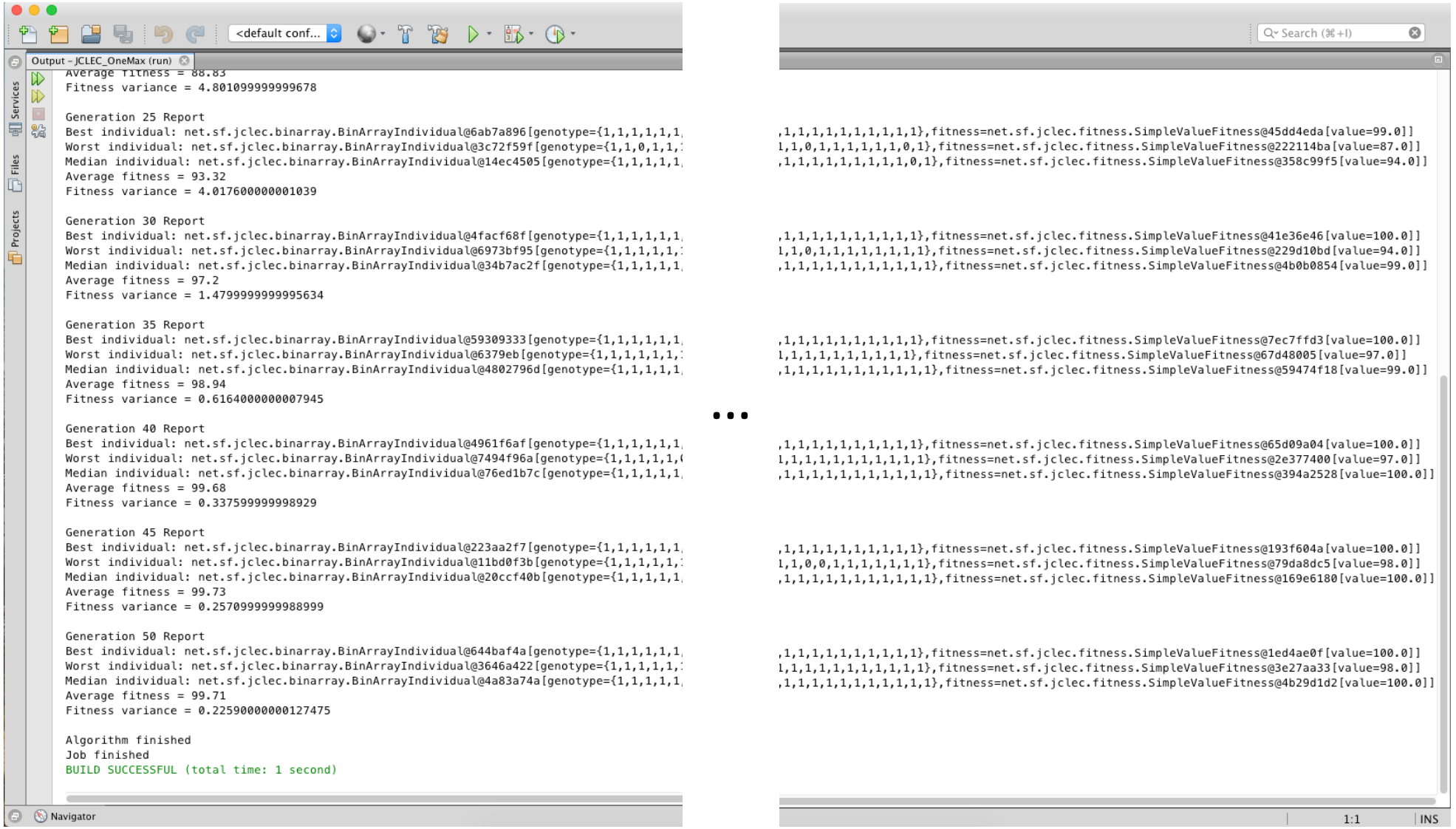
Here’s one possible configuration:

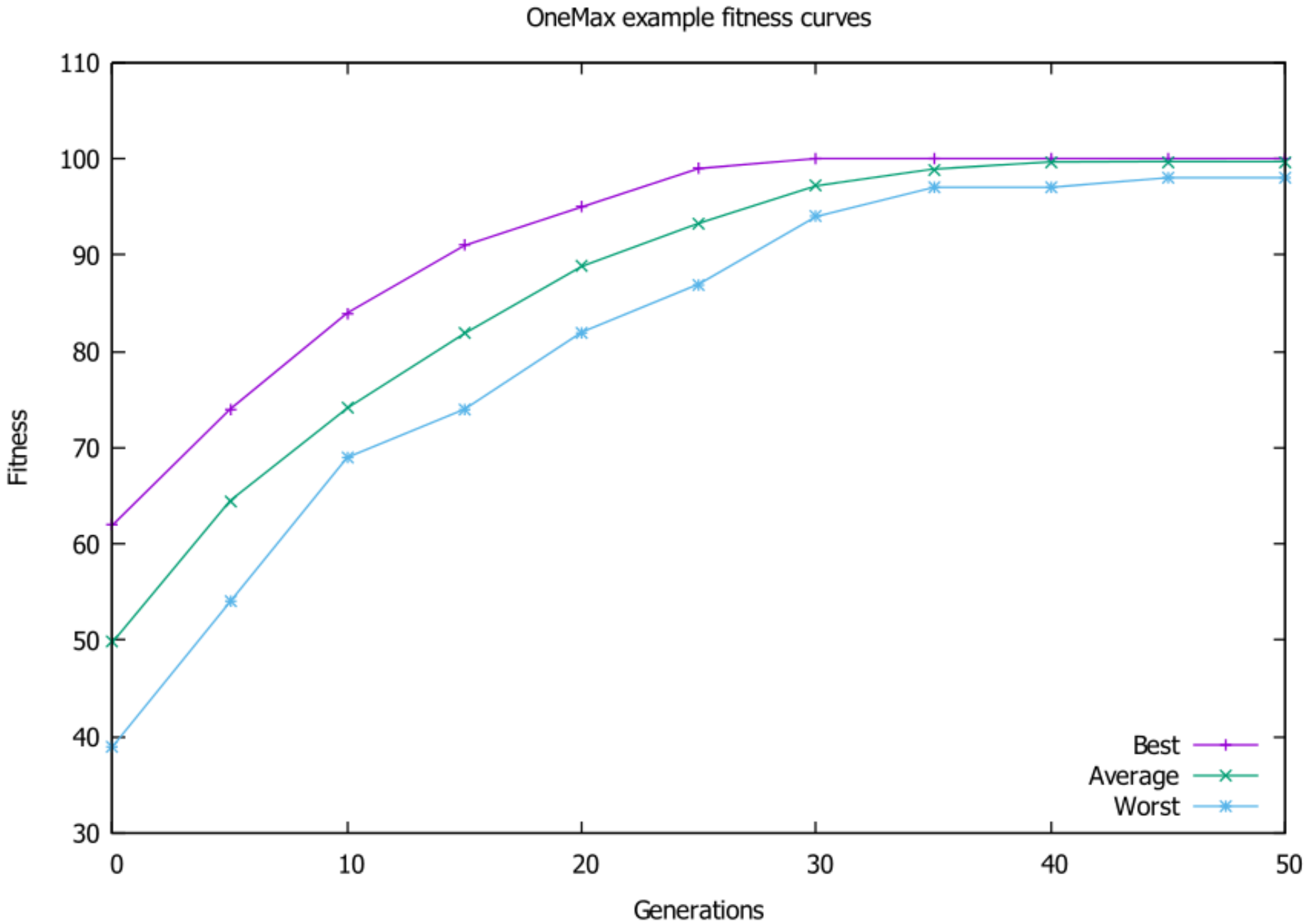
```
<experiment>
  <process algorithm-type="net.sf.jclec.algorithm.classic.SGE">
    <rand-gen-factory type="net.sf.jclec.util.random.RanecuFactory" seed="987328938"/>
    <population-size>100</population-size>
    <max-of-generations>50</max-of-generations>
    <species type="net.sf.jclec.binarray.BinArrayIndividualSpecies" genotype-length="100"/>
    <evaluator type="tutorial.OneMax"/>
    <provider type="net.sf.jclec.binarray.BinArrayCreator"/>
    <parents-selector type="net.sf.jclec.selector.TournamentSelector">
      <tournament-size>2</tournament-size>
    </parents-selector>
    <recombinator type="net.sf.jclec.binarray.rec.UniformCrossover" rec-prob="0.9" />
    <mutator type="net.sf.jclec.binarray.mut.OneLocusMutator" mut-prob="0.2" />
    <listener type="net.sf.jclec.listener.PopulationReporter">
      <report-frequency>5</report-frequency>
      <report-on-file>true</report-on-file>
      <save-complete-population>true</save-complete-population>
      <report-title>"OneMax-"</report-title>
   </listener>
  </process>
</experiment>
```

One way of solving the fitness evaluation:

```
@Override
protected void evaluate( IIndividual ind ) {
    // Individual genotype
    byte[ ] genotype = ( (BinArrayIndividual)ind).getGenotype( );
    int bitCount = 0;

    for( int i = 0; i < genotype.length; i++ )  {
        if( genotype[ i ] == 1 )  {
            bitCount++;
        }
    }

    ind.setFitness( new SimpleValueFitness( bitCount ) );
}
```

Demonstration


Output - JCLEC_OneMax (run)
Average fitness = 88.83
Fitness variance = 4.801099999999678
Generation 25 Report
Best individual: net.sf.jclec.binarray.BinArrayIndividual@6ab7a896[genotype={1,1,1,1,1,1,
Worst individual: net.sf.jclec.binarray.BinArrayIndividual@3c72f59f[genotype={1,1,0,1,1,
Median individual: net.sf.jclec.binarray.BinArrayIndividual@14ec4505[genotype={1,1,1,1,1,
Average fitness = 93.32
Fitness variance = 4.017600000001039
Generation 30 Report
Best individual: net.sf.jclec.binarray.BinArrayIndividual@4facf68f[genotype={1,1,1,1,1,1,
Worst individual: net.sf.jclec.binarray.BinArrayIndividual@6973bf95[genotype={1,1,1,1,1,
Median individual: net.sf.jclec.binarray.BinArrayIndividual@34b7ac2f[genotype={1,1,1,1,1,
Average fitness = 97.2
Fitness variance = 1.4799999999995634
Generation 35 Report
Best individual: net.sf.jclec.binarray.BinArrayIndividual@59309333[genotype={1,1,1,1,1,1,
Worst individual: net.sf.jclec.binarray.BinArrayIndividual@6379eb[genotype={1,1,1,1,1,1,
Median individual: net.sf.jclec.binarray.BinArrayIndividual@4802796d[genotype={1,1,1,1,1,
Average fitness = 98.94
Fitness variance = 0.6164000000007945
Generation 40 Report
Best individual: net.sf.jclec.binarray.BinArrayIndividual@4961f6af[genotype={1,1,1,1,1,1,
Worst individual: net.sf.jclec.binarray.BinArrayIndividual@7494f96a[genotype={1,1,1,1,1,
Median individual: net.sf.jclec.binarray.BinArrayIndividual@76ed1b7c[genotype={1,1,1,1,1,
Average fitness = 99.68
Fitness variance = 0.337599999998929
Generation 45 Report
Best individual: net.sf.jclec.binarray.BinArrayIndividual@223aa2f7[genotype={1,1,1,1,1,1,
Worst individual: net.sf.jclec.binarray.BinArrayIndividual@11bd0f3b[genotype={1,1,1,1,1,
Median individual: net.sf.jclec.binarray.BinArrayIndividual@20ccf40b[genotype={1,1,1,1,1,
Average fitness = 99.73
Fitness variance = 0.2570999999988999
Generation 50 Report
Best individual: net.sf.jclec.binarray.BinArrayIndividual@644baf4a[genotype={1,1,1,1,1,1,
Worst individual: net.sf.jclec.binarray.BinArrayIndividual@3646a422[genotype={1,1,1,1,1,
Median individual: net.sf.jclec.binarray.BinArrayIndividual@4a83a74a[genotype={1,1,1,1,1,
Average fitness = 99.71
Fitness variance = 0.22590000000127475
Algorithm finished
Job finished
BUILD SUCCESSFUL (total time: 1 second)
Navigator
Search (⌘+I)
,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@45dd4eda[value=99.0]]
l,1,0,1,1,1,1,1,1,0,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@222114ba[value=87.0]]
,1,1,1,1,1,1,1,1,1,0,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@358c99f5[value=94.0]]
,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@41e36e46[value=100.0]]
l,1,0,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@229d10bd[value=94.0]]
,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@4b0b0854[value=99.0]]
,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@7ec7ffd3[value=100.0]]
l,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@67d48005[value=97.0]]
,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@59474f18[value=99.0]]
,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@65d09a04[value=100.0]]
l,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@2e377400[value=97.0]]
,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@394a2528[value=100.0]]
,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@193f604a[value=100.0]]
l,1,0,0,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@79da8dc5[value=98.0]]
,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@169e6180[value=100.0]]
,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@1ed4ae0f[value=100.0]]
l,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@3e27aa33[value=98.0]]
,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@4b29d1d2[value=100.0]]
1:1
INS


OneMax example fitness curves
Fitness
Generations
110
100
90
80
70
60
50
40
30
0
10
20
30
40
50
Best
Average
Worst

Workshop

Evolutionary computing

Frameworks for evolutionary computing

Java Class Library for Evolutionary Computing (JCLEC)

Optimisation problems:

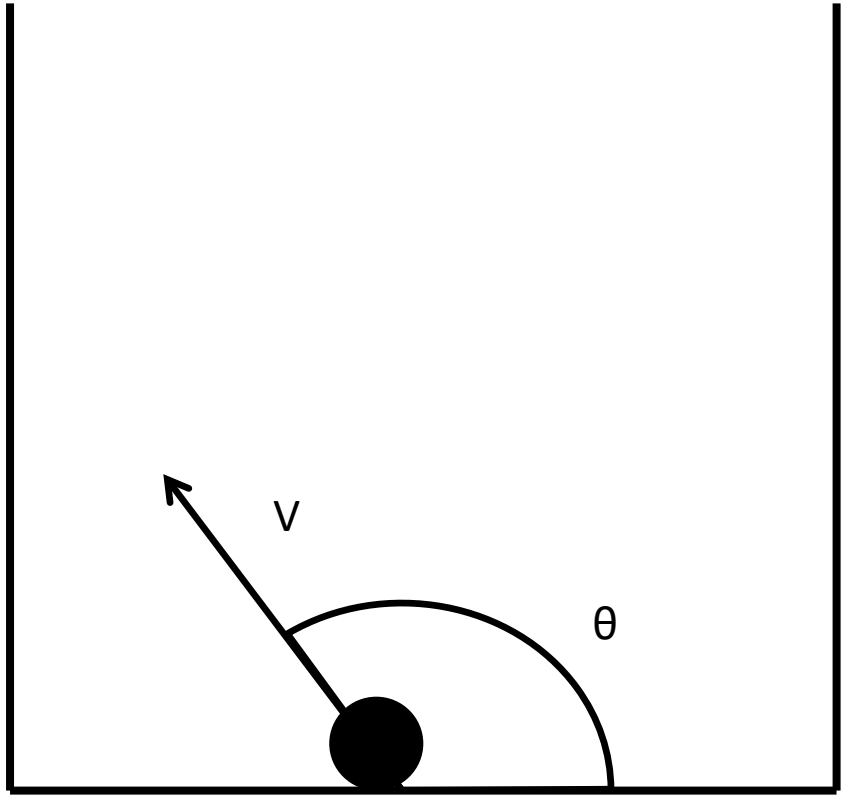
1 - ‘OneMax’ Problem

2 - How to program your way out of a paper bag

3 - FizzBuzz

Genetic Programming, Genetic Improvement

Let’s suppose there’s a canon in a paper bag



Let’s also suppose:
width of bag is 10.0,
height of bag is 5.0.

- *Overload*, 21(118):7–9, December 2013
  - http://accu.org/index.php/journals/1821

Given a bag with bottom left corner at (***k***, 0), of width ***w***, and height ***h***, assuming the projectile is smaller than the bag, the cannon is a point of no size, and given the acceleration due to gravity, ***g***, after time *t* the projectile will be at point (***x***, ***y***) where

$$x = k + \frac{1}{2}w + vt\cos\theta$$

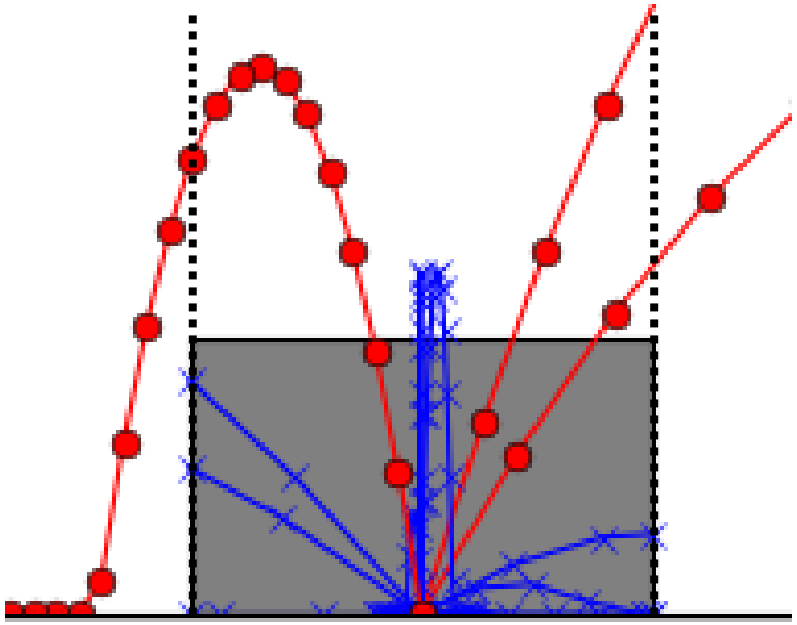
$$y = vt\sin\theta - \frac{1}{2}gt^2$$

x is the horizontal displacement and ***y*** the vertical displacement. The projectile will just escape when ***y*** ≥ ***h*** and ***x*** < ***k*** or ***x*** > ***k*** + ***w***.

g will be taken as 9.81 m/s^2. For simplicity, the code will assume ***k*** is zero.

Fitness evaluation

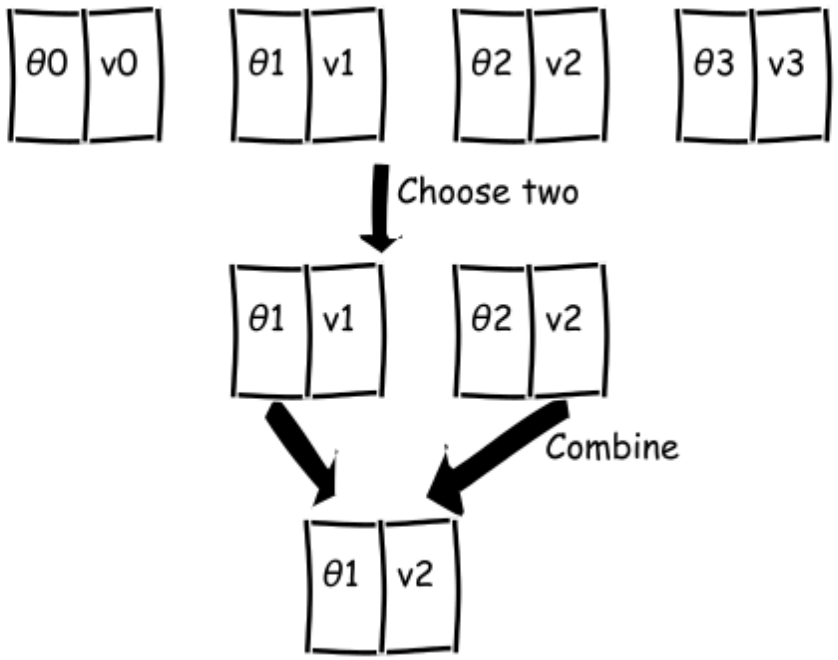
- Launching at random, something either ends in or out of the bag
- But some fail cases are less bad than others



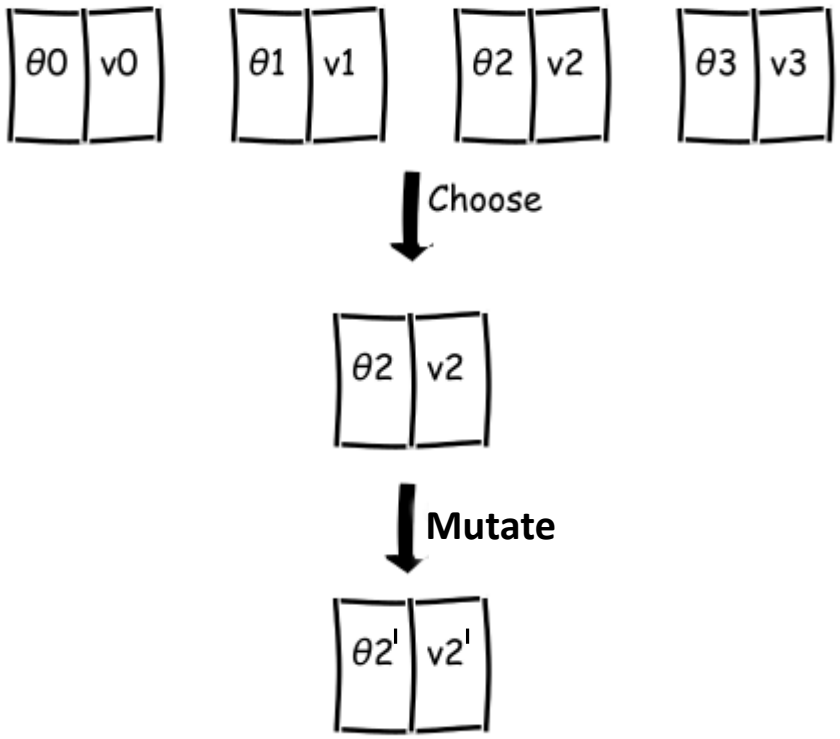
- 3 escape
- 2 on left get “close”
- Could “close” mean height (at edge of bag)?
  - Fitness = height

Diversity Preservation

Recombination



Mutation



Algorithm design and parameter set up – let's again apply some patterns…

Representation

- how to encode a candidate solution in the population?

?

Fitness

- how to evaluate the fitness of a candidate solution?

?

Diversity

- how to make offspring different to parents?

?

Initialisation: random?

Evolution: simple generational with elitism (SGE)?

… and suggested parameters

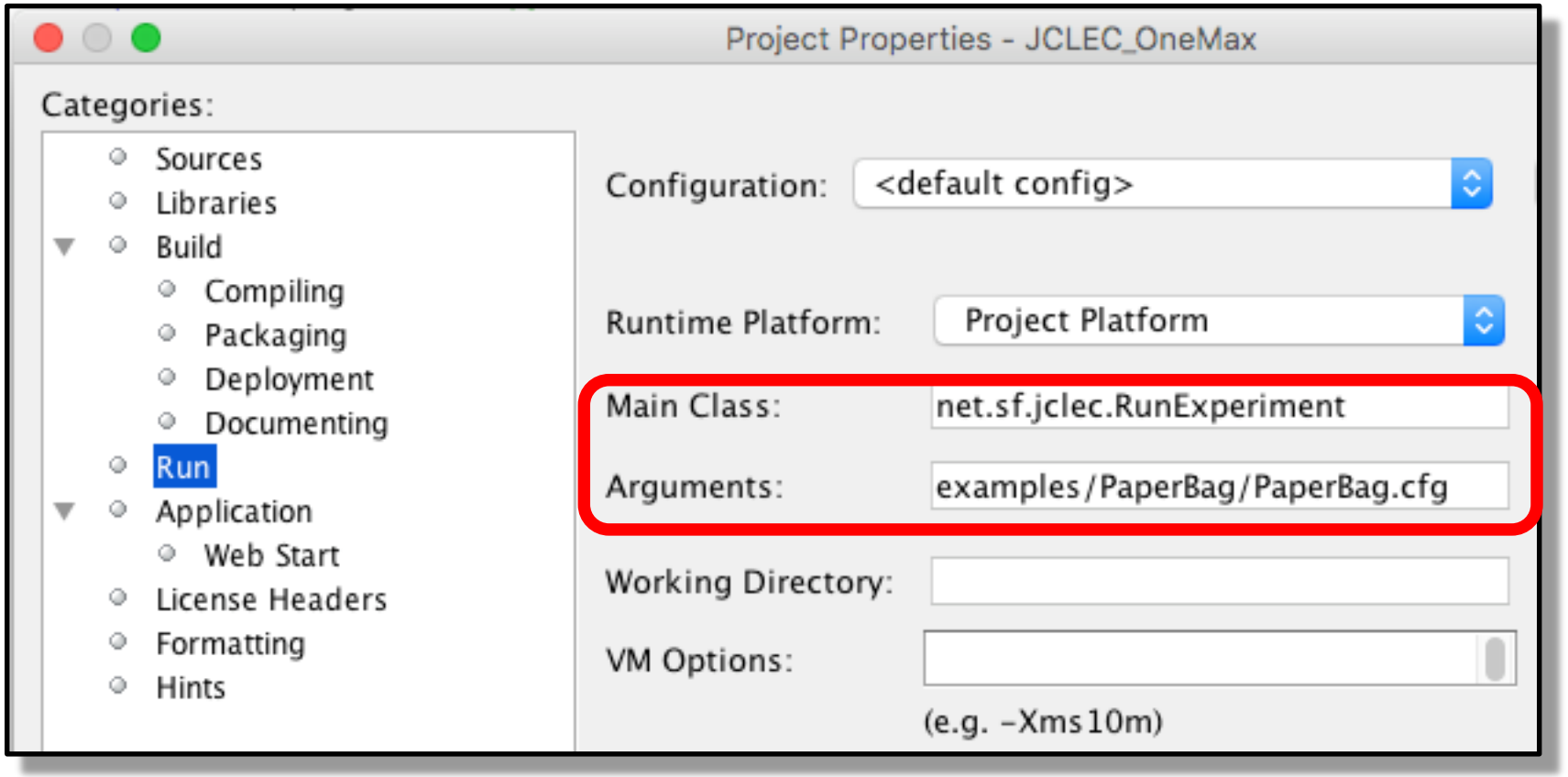
Population size: 12 individuals

Stop Criterion: 20 generations

Parent selection: tournament of 2 individuals

```
<experiment>
  <process algorithm-type="net.sf.jclec.algorithm.classic.SGE">
    <population-size>12</population-size>
    <max-of-generations>20</max-of-generations>
    <rand-gen-factory type="net.sf.jclec.util.random.RanecuFactory" seed="124321453"/>
    <species type="net.sf.jclec.realarray.RealArrayIndividualSpecies">
      <genotype-schema>
        <locus type="net.sf.jclec.util.range.Interval" left="0.0" right="20.0"
          closure="closed-closed"/>
        <locus type="net.sf.jclec.util.range.Interval" left="0.0" right="180.0"
          closure="closed-closed"/>
      </genotype-schema>
    </species>
    <evaluator type="tutorial.PaperBag"/>
    <provider type="net.sf.jclec.realarray.RealArrayCreator"/>
    <parents-selector type="net.sf.jclec.selector.TournamentSelector"
      tournament-size="2"/>
    <mutator type="net.sf.jclec.realarray.mut.NonUniformMutator" mut-prob="0.15" />
    <recombinator type="net.sf.jclec.realarray.rec.BLXAlphaCrossover" rec-prob="0.9"
      alpha="0.3"/>
    <listener … </listener>
  </process>
</experiment>
```

Don’t forget to invoke the executable with ”PaperBag.cfg” as an argument



Enjoy the programming!

Hint – think about converting the angle theta to radians before applying `sin()` and `cos()`

Here's one way to solve the 'Out of a Paper Bag' optimisation problem…

Let’s start with a ‘Point’ class (with public x & y attributes for convenience):

```
public class Point  {
    public double x;
    public double y;

    public Point( )  {
        x = 0.0;
        y = 0.0;
    }
}
```

```
public class PaperBag extends AbstractEvaluator {

   protected boolean maximize = true;

   private Comparator<IFitness> COMPARATOR;

   /* list of x,y points of the projectile trajectory */
   private List< Point > pointsList;

   private DecimalFormat df;  // for debugging

   private static final double GRAVITY = 9.81; // gravity i.e. 9.81 m/sec²

   private static final double WIDTH = 10.0; // width of the paper bag

   private static final double HEIGHT = 5.0; // height of the paper bag

   private static final double STEP = 0.1; // the "time step"

   // …
```

Continued….

One way of solving the fitness evaluation:

```
protected void evaluate( IIndividual ind ) {
    // Individual genotype
    double[ ] genotype = ((RealArrayIndividual)ind).getGenotype( );
    double velocity = genotype[ 0 ];
    double theta = genotype[ 1 ];

    pointsList = new ArrayList< >( ); // clear out the list of points

    // calculate the points of the parabolic trajectory
    for( double time = 0.0; time < END; time += STEP ){
        Point p = getPointAtTime( time, velocity, theta );
        pointsList.add( p );
    }

    // …
```

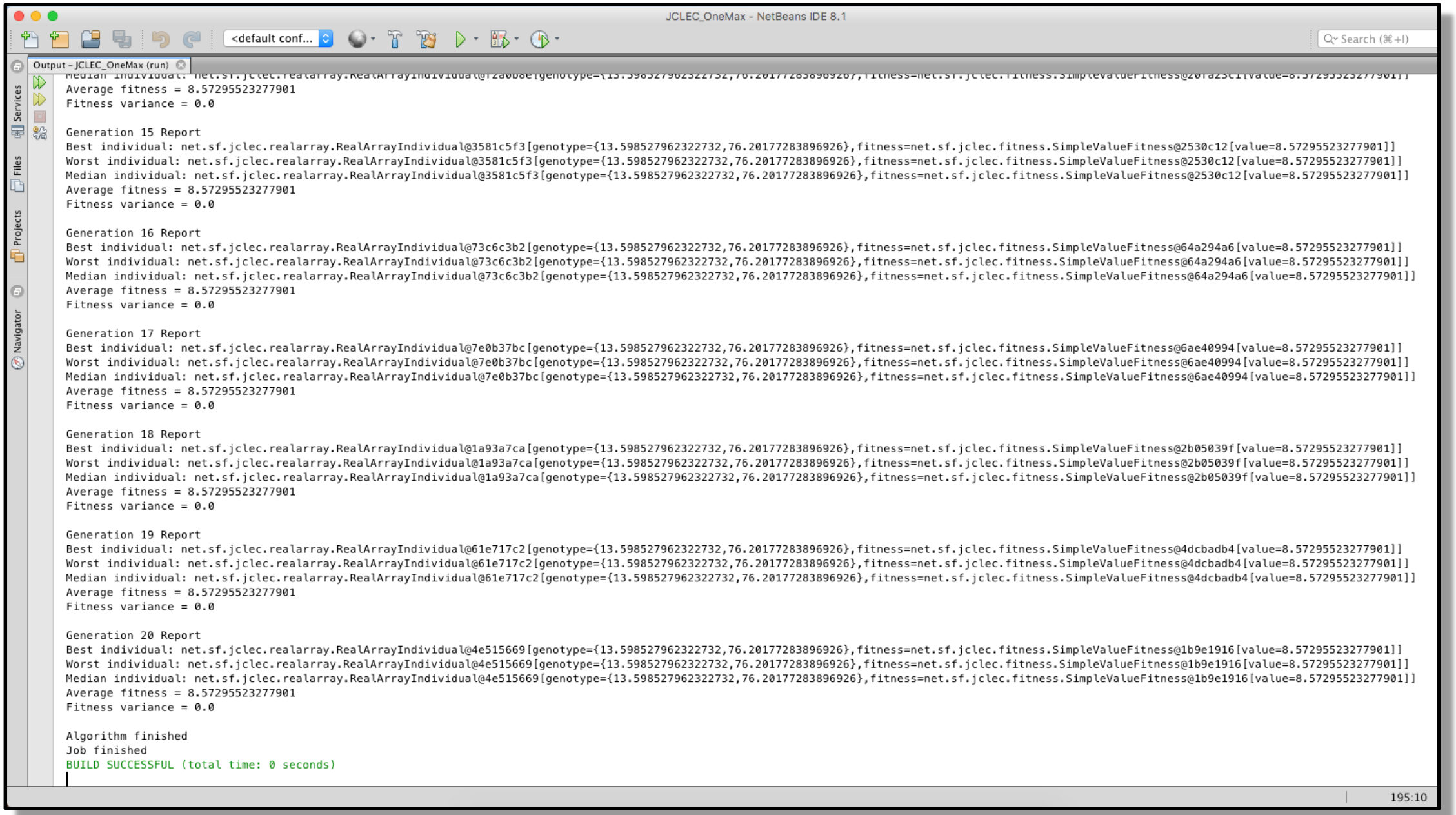
Continued….

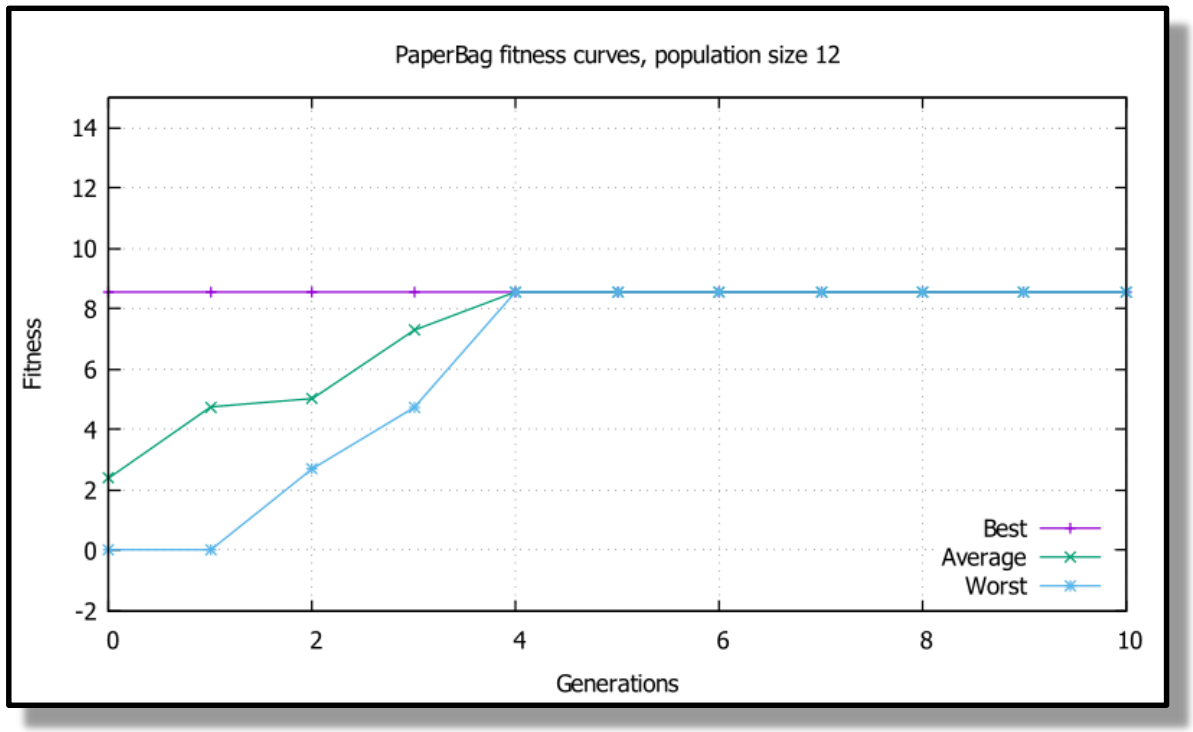
```
        double fitness = 0.0;

        // calculate fitness value from the parabolic trajectory points
        for( Point p : pointsList ) {
            if( p.x <= 0.0 || p.x >= WIDTH ) {
                fitness = p.y;
                break;
            }
        }

        ind.setFitness( new SimpleValueFitness( fitness ) );
    }
```

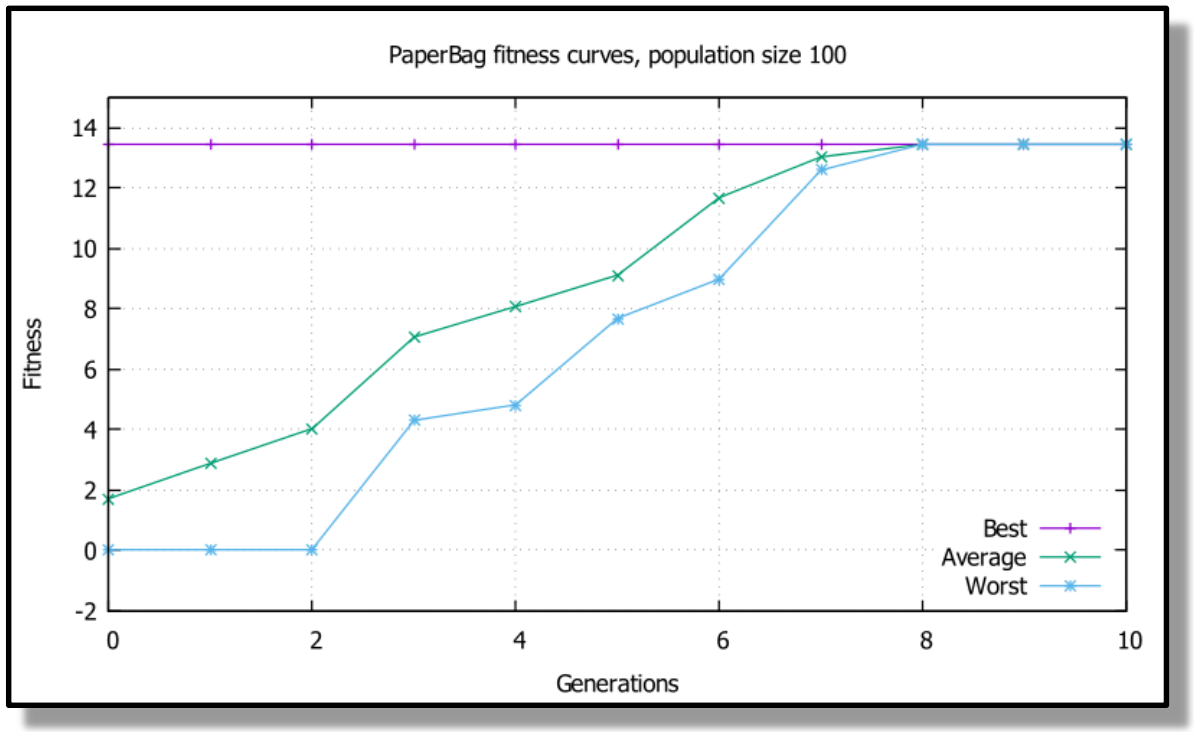
The getPointAtTime( ) method:

```
private Point getPointAtTime( final double time, final double velocity,
    final double theta )     {
    double inRadians = Math.toRadians( theta ); // convert to radians

    double xTemp = 0.5 * WIDTH;
    double xIncrement = velocity * time * Math.cos( inRadians );
    xTemp += xIncrement;

    double yTemp = velocity * time * Math.sin( inRadians );
    double yIncrement = 0.5 * GRAVITY * ( time * time );
    yTemp -= yIncrement;
    // can't have a negative y value - this is the ground!
    if( yTemp < 0.0 ) yTemp = 0.0;

    Point p = new Point( );
    p.x = xTemp; p.y = yTemp;
    return p;
}
```

Demonstration


JCLEC_OneMax - NetBeans IDE 8.1
<default conf...
Search (⌘+I)
Output - JCLEC_OneMax (run)
Services
Files
Projects
Navigator
Average fitness = 8.57295523277901
Fitness variance = 0.0
Generation 15 Report
Best individual: net.sf.jclec.realarray.RealArrayIndividual@3581c5f3[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@2530c12[value=8.57295523277901]]
Worst individual: net.sf.jclec.realarray.RealArrayIndividual@3581c5f3[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@2530c12[value=8.57295523277901]]
Median individual: net.sf.jclec.realarray.RealArrayIndividual@3581c5f3[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@2530c12[value=8.57295523277901]]
Average fitness = 8.57295523277901
Fitness variance = 0.0
Generation 16 Report
Best individual: net.sf.jclec.realarray.RealArrayIndividual@73c6c3b2[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@64a294a6[value=8.57295523277901]]
Worst individual: net.sf.jclec.realarray.RealArrayIndividual@73c6c3b2[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@64a294a6[value=8.57295523277901]]
Median individual: net.sf.jclec.realarray.RealArrayIndividual@73c6c3b2[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@64a294a6[value=8.57295523277901]]
Average fitness = 8.57295523277901
Fitness variance = 0.0
Generation 17 Report
Best individual: net.sf.jclec.realarray.RealArrayIndividual@7e0b37bc[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@6ae40994[value=8.57295523277901]]
Worst individual: net.sf.jclec.realarray.RealArrayIndividual@7e0b37bc[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@6ae40994[value=8.57295523277901]]
Median individual: net.sf.jclec.realarray.RealArrayIndividual@7e0b37bc[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@6ae40994[value=8.57295523277901]]
Average fitness = 8.57295523277901
Fitness variance = 0.0
Generation 18 Report
Best individual: net.sf.jclec.realarray.RealArrayIndividual@1a93a7ca[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@2b05039f[value=8.57295523277901]]
Worst individual: net.sf.jclec.realarray.RealArrayIndividual@1a93a7ca[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@2b05039f[value=8.57295523277901]]
Median individual: net.sf.jclec.realarray.RealArrayIndividual@1a93a7ca[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@2b05039f[value=8.57295523277901]]
Average fitness = 8.57295523277901
Fitness variance = 0.0
Generation 19 Report
Best individual: net.sf.jclec.realarray.RealArrayIndividual@61e717c2[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@4dcbadb4[value=8.57295523277901]]
Worst individual: net.sf.jclec.realarray.RealArrayIndividual@61e717c2[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@4dcbadb4[value=8.57295523277901]]
Median individual: net.sf.jclec.realarray.RealArrayIndividual@61e717c2[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@4dcbadb4[value=8.57295523277901]]
Average fitness = 8.57295523277901
Fitness variance = 0.0
Generation 20 Report
Best individual: net.sf.jclec.realarray.RealArrayIndividual@4e515669[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@1b9e1916[value=8.57295523277901]]
Worst individual: net.sf.jclec.realarray.RealArrayIndividual@4e515669[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@1b9e1916[value=8.57295523277901]]
Median individual: net.sf.jclec.realarray.RealArrayIndividual@4e515669[genotype={13.598527962322732,76.20177283896926},fitness=net.sf.jclec.fitness.SimpleValueFitness@1b9e1916[value=8.57295523277901]]
Average fitness = 8.57295523277901
Fitness variance = 0.0
Algorithm finished
Job finished
BUILD SUCCESSFUL (total time: 0 seconds)
195:10


PaperBag fitness curves, population size 12
Fitness
Generations
Best
Average
Worst
14
12
10
8
6
4
2
0
-2
0
2
4
6
8
10


PaperBag fitness curves, population size 100
Fitness
Generations
Best
Average
Worst
14
12
10
8
6
4
2
0
-2
0
2
4
6
8
10

Workshop

Evolutionary computing

Frameworks for evolutionary computing

Java Class Library for Evolutionary Computing (JCLEC)

Optimisation problems:

1 - ‘OneMax’ Problem

2 - How to program your way out of a paper bag

3 - FizzBuzz

Genetic Programming, Genetic Improvement

Rules of FizzBuzz

A typical game of FizzBuzz involves counting through a sequence of numbers starting at one, but multiples of three are substituted with ‘Fizz’, multiples of five are substituted with ‘Buzz’, and multiples of fifteen are substituted with ‘FizzBuzz’.

Let’s start by evolving a sequence of 100 integers…

- First define the actual sequence in the configure method in a `FizzBuzz` class
  - Here are some constants that might be useful…

```
private List< Integer > sequence; // to evolve against

private static final int SEQUENCE_SIZE = 100;

private static final int FIZZ = 3;
private static final int BUZZ = 5;
private static final int FIZZ_BUZZ = 15;

private static final Integer FIZZ_IDENTIFIER = 101;
private static final Integer BUZZ_IDENTIFIER = 102;
private static final Integer FIZZ_BUZZ_IDENTIFIER = 103;
```

Enjoy the programming!

Here’s one way to solve the ‘FizzBuzz as a sequence’ problem…

```
public void configure(Configuration settings)  {
    sequence = new ArrayList< >( );

    for( int i = 1; i <= SEQUENCE_SIZE; i++ ) {
        sequence.add( new Integer( i ) );
    }

    for( int i = 1; i < sequence.size( ); i++ ) {
        int number = sequence.get( i );
        if( number % FIZZ_BUZZ == 0 ) {
            sequence.set( i, new Integer( FIZZ_BUZZ_IDENTIFIER ) );
        }
        else if( number % BUZZ == 0 ) {
            sequence.set( i, new Integer( BUZZ_IDENTIFIER ) );
        }
        else if( number % FIZZ == 0 ) {
            sequence.set( i, new Integer( FIZZ_IDENTIFIER ) );
        }
    }
}
```

Here's the fizzbuzz.cfg file (1 of 2):

```
<experiment>
    <process algorithm-type="net.sf.jclec.algorithm.classic.SGE">
        <rand-gen-factory type="net.sf.jclec.util.random.RanecuFactory"
            seed="123456789"/>
        <population-size>100</population-size>
        <max-of-generations>3000</max-of-generations>

        <provider type="net.sf.jclec.intarray.IntArrayCreator" />
             <species type="net.sf.jclec.intarray.IntArrayIndividualSpecies"
                 genotype-length="100">
                <genotype-schema>
                    <locus type="net.sf.jclec.util.intset.Interval"
                        left="1" right="103" closure="closed-closed" />
                    <locus type="net.sf.jclec.util.intset.Interval"
                        left="1" right="103" closure="closed-closed" />

                     ... Etc. for each integer in the array of 100 integers ...

                </genotype-schema>
            </species>
```

Continued…

Here's the fizzbuzz.cfg file (2 of 2):

```
    <evaluator type="tutorial.FizzBuzz"> </evaluator>

    <parents-selector type="net.sf.jclec.selector.TournamentSelector">
                <tournament-size>2</tournament-size>
    </parents-selector>

    <mutator type="net.sf.jclec.intarray.mut.OneLocusMutator" mut-prob="0.2" />
    <recombinator type="net.sf.jclec.intarray.rec.OnePointCrossover" rec-prob="0.9"/>

    <listener type="net.sf.jclec.listener.PopulationReporter">
          <report-frequency>10</report-frequency>
          <report-on-file>false</report-on-file>
          <save-complete-population>false</save-complete-population>
          <report-title>FizzBuzz-</report-title>
    </listener>
  </process>
</experiment>
```

The evaluation( ) method is almost trivial – something like ‘OneMax’?

```
protected void evaluate( IIndividual ind ) {
    int [] genotype = ( ( IntArrayIndividual ) ind ).getGenotype( );

    int matchCount = 0;

    for( int i = 0; i < genotype.length; i++ ) {
        if( genotype[ i ] == sequence.get( i ) ) {
            matchCount++;
        }
    }

    ind.setFitness( new SimpleValueFitness( matchCount ) );
}
```

Demonstration

But there’s a problem with this approach….

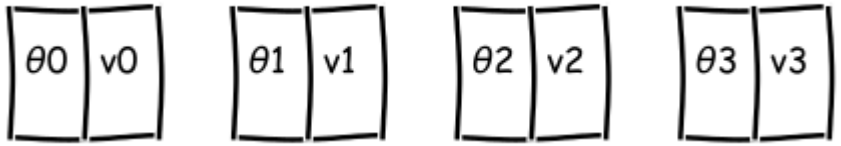
- How do you write the fitness function without writing code to generate the correct values?
- What properties do we need?
- Listen to @KevlinHenney

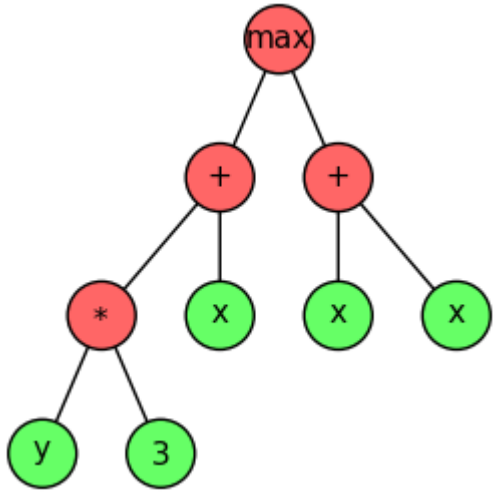
every result is 'Fizz', 'Buzz', 'FizzBuzz' or a decimal string,
every decimal result corresponds to its ordinal position,
every third result contains 'Fizz',
every fifth result contains 'Buzz',
every fifteenth result is 'FizzBuzz',
the ordinal position of every 'Fizz' result is divisible by 3,
the ordinal position of every 'Buzz' result is divisible by 5,
the ordinal position of every 'FizzBuzz' result is divisible by 15

```
actual = [fizzbuzz(n) for n in range(1, 101)]
truths = [
  all(a in {'Fizz', 'Buzz', 'FizzBuzz'} or a.isdecimal() for a in actual),
  all(int(a) == n for n, a in enumerate(actual, 1) if a.isdecimal()),
  all('Fizz' in a for a in actual[2::3]),
  all('Buzz' in a for a in actual[4::5]),
  all(a == 'FizzBuzz' for a in actual[14::15]),
  all(n % 3 == 0 for n, a in enumerate(actual, 1) if a == 'Fizz'),
  all(n % 5 == 0 for n, a in enumerate(actual, 1) if a == 'Buzz'),
  all(n % 15 == 0 for n, a in enumerate(actual, 1) if a == 'FizzBuzz')
]
assert all(truths)
```

Change the array to a tree

- Previous genotypes were “list/array”
- What if we use trees?
- What do trees make?
- Code

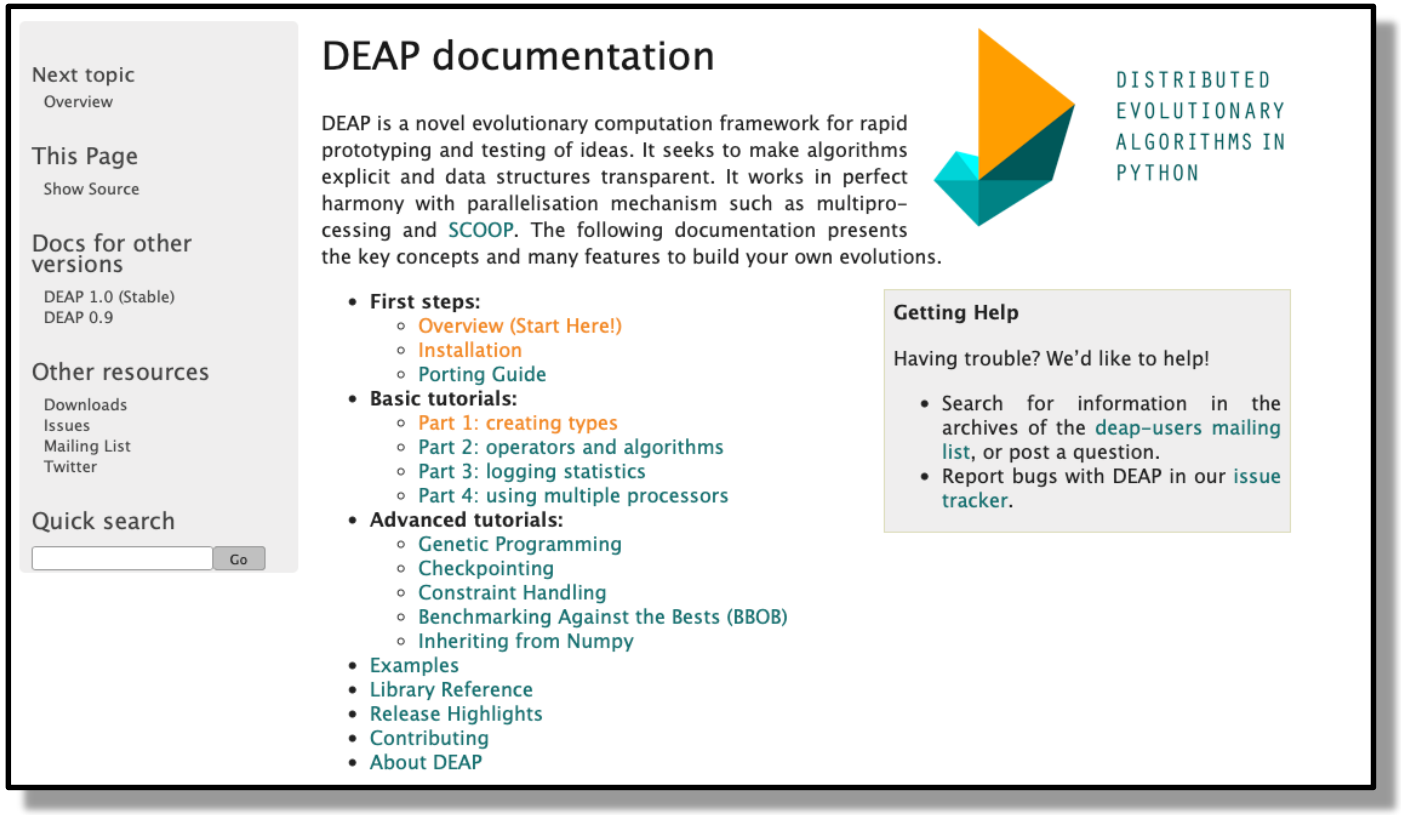




But JCLEC doesn't specifically offer AST representations… and that looked a bit like Python…

Distributed Evolutionary Algorithms in Python (DEAP)

https://deap.readthedocs.io/en/master/



So let’s evolve a tree

https://deap.readthedocs.io/en/master/tutorials/advanced/gp.html

- You add a primitive set,
```
pset = PrimitiveSet("main", 2)
```
- Add function and number operators:
```
pset.addPrimitive(max, 2)
pset.addPrimitive(operator.add, 2)
pset.addPrimitive(operator.neg, 1)
```
- Or values:
```
pset.addTerminal(3)
```
- Then generate trees:
```
expr = genFull(pset, min_=1, max_=3)
tree = PrimitiveTree(expr)
```

For Fizz Buzz

```
def if_then_else(x,y,z):
    if x:
        return y
    else:
        return z

def mod3(x):
    return operator.mod(x, 3) == 0

def mod5(x):
    return operator.mod(x, 5) == 0

def mod15(x):
    return operator.mod(x, 15) == 0

def both(x, y):
    return x and y

def either(x, y):
    return x or y
```

```
pset = gp.PrimitiveSet("MAIN", 1)

pset.addPrimitive(operator.add, 2)
pset.addPrimitive(operator.sub, 2)
pset.addPrimitive(operator.mul, 2)

pset.addPrimitive(both, 2)
pset.addPrimitive(either, 2)

pset.addPrimitive(operator.mod, 2)

pset.addPrimitive(if_then_else, 3)
pset.addPrimitive(mod3, 1)
pset.addPrimitive(mod5, 1)
pset.addPrimitive(mod15, 1)


pset.addTerminal("Buzz")
pset.addTerminal("Fizz")
pset.addTerminal("FizzBuzz")
```

Can get it to find numbers, e.g.

```
pset.addEphemeralConstant("rand101", lambda: random.randint(-1,1))
```

But don’t have a working example yet

Evolve…

```
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", gp.PrimitiveTree,
               fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("expr", gp.genHalfAndHalf,
                 pset=pset, min_=1, max_=2)
toolbox.register("individual", tools.initIterate,
                 creator.Individual, toolbox.expr)
toolbox.register("population", tools.initRepeat, list,
                 toolbox.individual)
toolbox.register("compile", gp.compile, pset=pset)
```

```
#or genFull or genGrow
# genHalfAndHalf does grow 50% or
# time, Full 50%
```

Fitness…

```
def fizz_buzz(func, points):
    passed = 0
    def safe_run(func, x):
        try:
            return func(x)
        except:
            return -1
    results = [safe_run(func, x) for x in points
    if every_result_is_Fizz_Buzz_FizzBuzz_or_a_decimal(results):
        passed += 1
    if every_decimal_result_corresponds_to_its_ordinal_position(results):
        passed += 1

    if every_third_result_contains_Fizz(results):
        passed += 1
    if every_fifth_result_contains_Buzz(results):
        passed += 1

    if every_fifteenth_result_contains_FizzBuzz(results):
        passed += 1
    if the_ordinal_position_of_every_Fizz_result_is_divisible_by_3(results):
        passed += 1
    if the_ordinal_position_of_every_Buzz_result_is_divisible_by_5(results):
        passed += 1
    if the_ordinal_position_of_every_FizzBuzz_result_is_divisible_by_15(results):
        passed += 1
    return passed
```

```
def register(fn):    #e.g. our fizz buzz tests fitness function

    def eval(individual, points): # This is our custom evaluation function
        # Transform the tree expression in a callable function
        func = toolbox.compile(expr=individual)
       # Evaluate the result, somehow
       return fn(func, points),

    toolbox.register("evaluate", eval, points=range(101)) # and we register it here
    toolbox.register("select", tools.selTournament, tournsize=3)
    toolbox.register("mate", gp.cxOnePoint)
    toolbox.register("expr_mut", gp.genFull, min_=0, max_=2)
    toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr_mut, pset=pset)

    toolbox.decorate("mate", gp.staticLimit(key=operator.attrgetter("height"), max_value=17))
    toolbox.decorate("mutate", gp.staticLimit(key=operator.attrgetter("height"), max_value=17))

def main():
    random.seed(318)
    register(fizz_buzz)
    pop = toolbox.population(n=4000)
    hof = tools.HallOfFame(1) #The best

    pCrossover = 0.75
    pMutation = 0.5
    nGen = 75
    pop, log = algorithms.eaSimple(pop, toolbox, pCrossover, pMutation, nGen, halloffame=hof)
#can add other stats
    return pop, log, hof
```

Ta-da!

```
['FizzBuzz', 1, 2, 'Fizz', 4, 'Buzz', 'Fizz', 7, 8, 'Fizz', 'Buzz', 11, 'Fizz', 13, 14, 'FizzBuzz', 16, 17, 'Fizz', 19,
'Buzz', 'Fizz', 22, 23, 'Fizz', 'Buzz', 26, 'Fizz', 28, 29, 'FizzBuzz', 31, 32, 'Fizz', 34, 'Buzz', 'Fizz', 37, 38, 'Fiz
z', 'Buzz', 41, 'Fizz', 43, 44, 'FizzBuzz', 46, 47, 'Fizz', 49, 'Buzz', 'Fizz', 52, 53, 'Fizz', 'Buzz', 56, 'Fizz', 58,
59, 'FizzBuzz', 61, 62, 'Fizz', 64, 'Buzz', 'Fizz', 67, 68, 'Fizz', 'Buzz', 71, 'Fizz', 73, 74, 'FizzBuzz', 76, 77, 'Fiz
z', 79, 'Buzz', 'Fizz', 82, 83, 'Fizz', 'Buzz', 86, 'Fizz', 88, 89, 'FizzBuzz', 91, 92, 'Fizz', 94, 'Buzz', 'Fizz', 97,
98, 'Fizz', 'Buzz']
```

The Hof

```
if_then_else(mod15(if_then_else(if_then_else(mul(x, 'FizzBuzz'), 'Fizz',
'Buzz'), x, if_then_else('Buzz', 'FizzBuzz', mod3(x)))), 'FizzBuzz',
if_then_else(both(if_then_else(if_then_else(mod15(x),
either('FizzBuzz', 'FizzBuzz'), 'FizzBuzz'), if_then_else('FizzBuzz',
mod15(mod5(x)), 'Buzz'), 'Buzz'), if_then_else('Fizz', 'Buzz',
if_then_else('FizzBuzz', if_then_else(if_then_else('Buzz',
if_then_else(if_then_else(mod3(x), x, 'FizzBuzz'), if_then_else(x, x,
either('Buzz', 'Buzz')), x), 'Fizz'), 'Fizz', x),
if_then_else(either(if_then_else(x, x, mod3(x)), 'FizzBuzz'), 'Fizz',
'Fizz')))), if_then_else(mod15(x), either('FizzBuzz', either('Buzz', x)),
if_then_else(mod3(x), 'Fizz', x)), 'Buzz'))
```

The Hof

```
if_then_else(mod15(if_then_else(if_then_else(mul(x, 'FizzBuzz'), 'Fizz',
'Buzz'), x, if_th                                    izzBuzz',
if_then_else(b                                       x),
either('FizzBuz                                      izzBuzz',
mod15(mod5(x                                         Buzz',
if_then_else('F                                      zz',
if_then_else(if_                                     hen_else(x, x,
either('Buzz', 'B
if_then_else(either(if_then_else(x, x, mod3(x)), 'FizzBuzz'), 'Fizz',
'Fizz'))))), if_then_else(mod15(x), either('FizzBuzz', either('Buzz', x)),
if_then_else(mod3(x), 'Fizz', x)), 'Buzz'))
```

Workshop

Evolutionary computing

Frameworks for evolutionary computing

Java Class Library for Evolutionary Computing (JCLEC)

Optimisation problems:

1 - ‘OneMax’ Problem

2 - How to program your way out of a paper bag

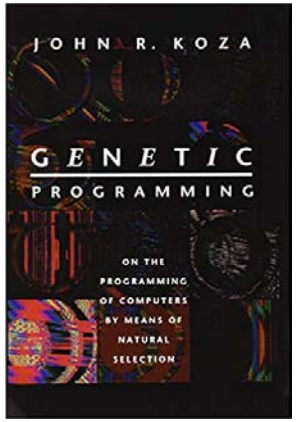
3 - FizzBuzz

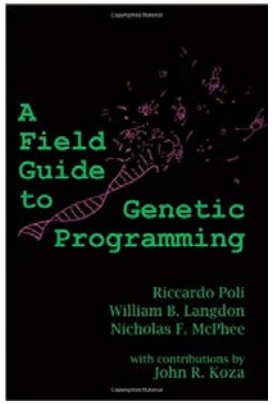
Genetic Programming, Genetic Improvement

Genetic Programming (GP) - evolution of a tree structure

Evolves the 'innards' (white box) of a function or expression

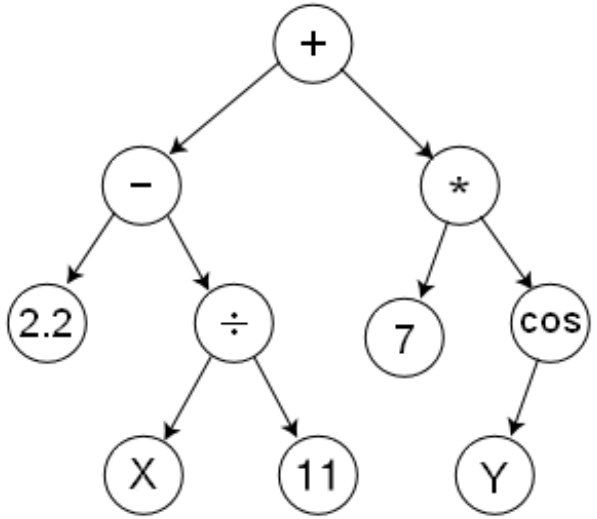
Each tree node is an *operator* or *variable*,
or a *terminal node*.





Used widely to evolve functions for

- Curve fitting
- Circuit board design
- Data modelling
- Symbolic regression
- Feature selection
- Classification



$$\left(2.2 - \left(\frac{X}{11}\right)\right) + \left(7 * \cos(Y)\right)$$

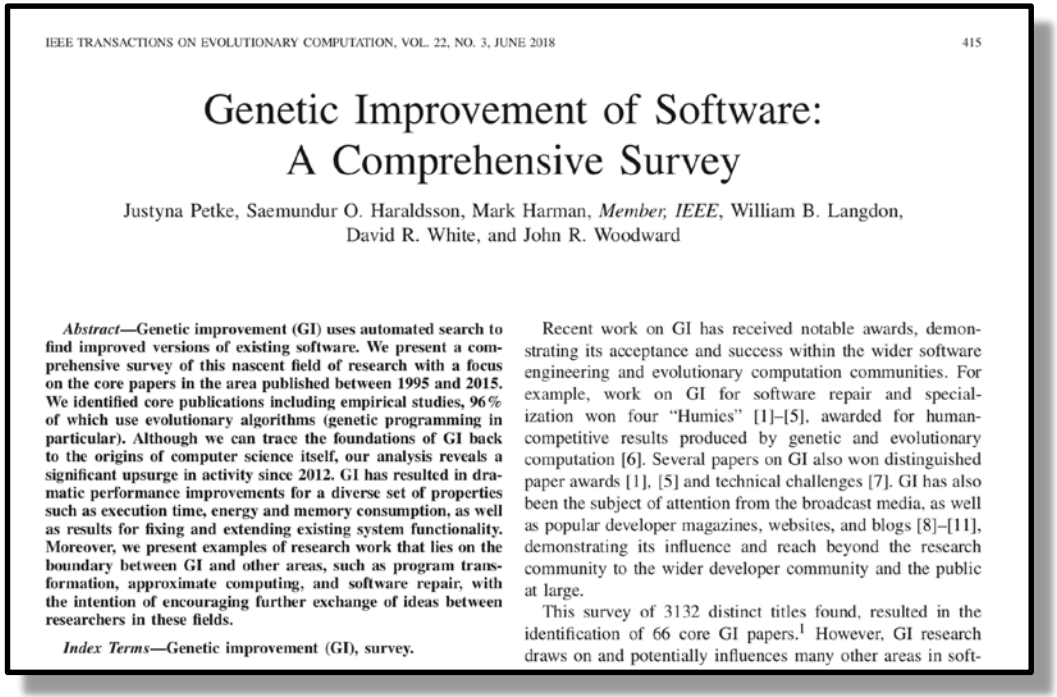
https://en.wikipedia.org/wiki/Genetic_programming#/media/File:Genetic_Program_Tree.png

Could we use GP to evolve source code e.g. for FizzBuzz?

Well, using AST representations, as we saw above, yes!

BUT GP doesn't always scale well (e.g. due to 'code bloat' with crossover) and *code doesn't look like what a human programmer would produce!!!!*

So in practice, maybe better to start with an existing body of code and use ***'Genetic Improvement' (GI)*** e.g.



IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 22, NO. 3, JUNE 2018 415

Genetic Improvement of Software: A Comprehensive Survey

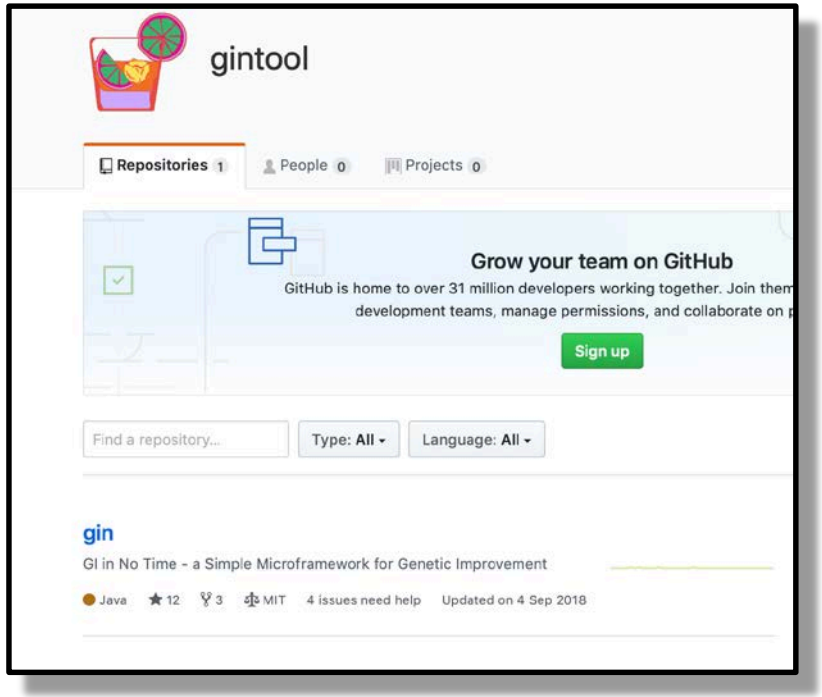
Justyna Petke, Saemundur O. Haraldsson, Mark Harman, *Member, IEEE*, William B. Langdon, David R. White, and John R. Woodward

***Abstract*—Genetic improvement (GI) uses automated search to find improved versions of existing software. We present a comprehensive survey of this nascent field of research with a focus on the core papers in the area published between 1995 and 2015. We identified core publications including empirical studies, 96% of which use evolutionary algorithms (genetic programming in particular). Although we can trace the foundations of GI back to the origins of computer science itself, our analysis reveals a significant upsurge in activity since 2012. GI has resulted in dramatic performance improvements for a diverse set of properties such as execution time, energy and memory consumption, as well as results for fixing and extending existing system functionality. Moreover, we present examples of research work that lies on the boundary between GI and other areas, such as program transformation, approximate computing, and software repair, with the intention of encouraging further exchange of ideas between researchers in these fields.**

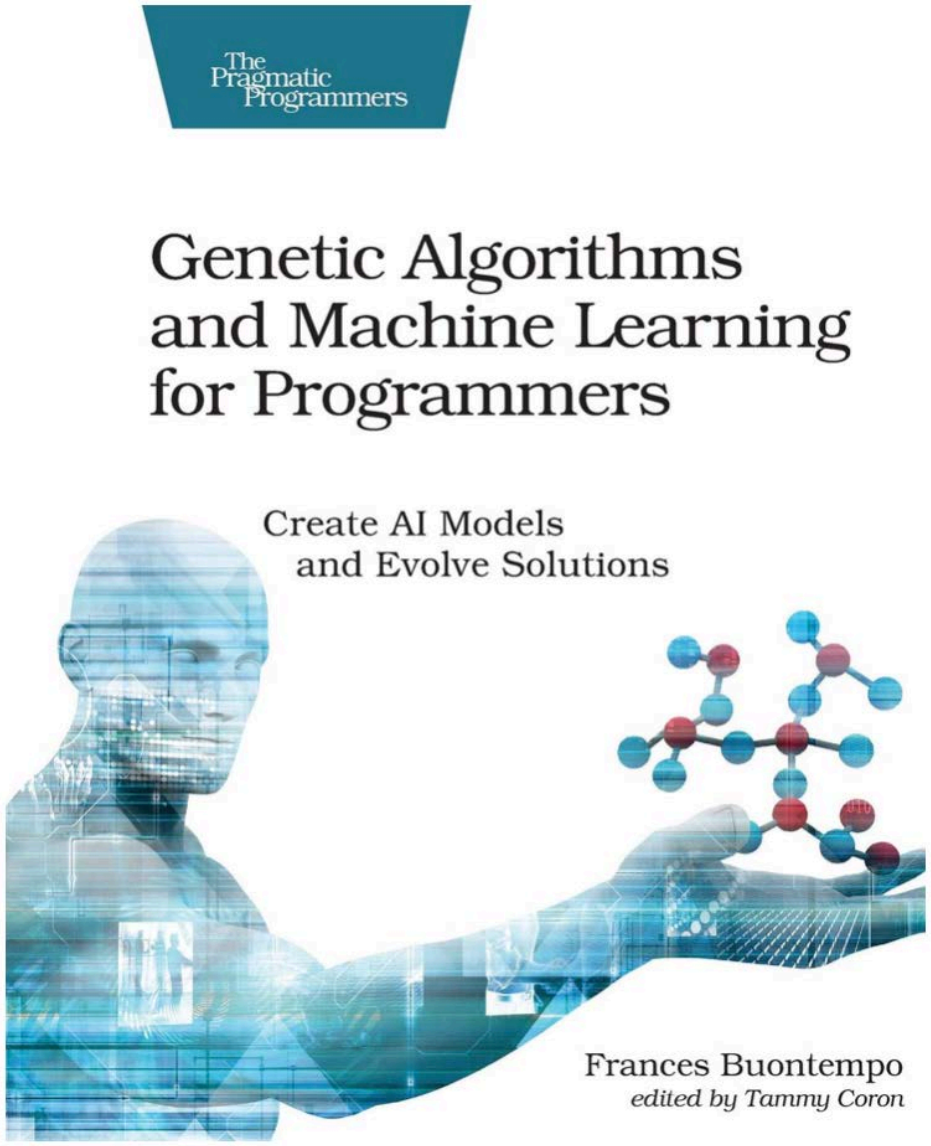
***Index Terms*—Genetic improvement (GI), survey.**

Recent work on GI has received notable awards, demonstrating its acceptance and success within the wider software engineering and evolutionary computation communities. For example, work on GI for software repair and specialization won four "Humies" [1]–[5], awarded for human-competitive results produced by genetic and evolutionary computation [6]. Several papers on GI also won distinguished paper awards [1], [5] and technical challenges [7]. GI has also been the subject of attention from the broadcast media, as well as popular developer magazines, websites, and blogs [8]–[11], demonstrating its influence and reach beyond the research community to the wider developer community and the public at large.

This survey of 3132 distinct titles found, resulted in the identification of 66 core GI papers.[1] However, GI research draws on and potentially influences many other areas in soft-



For further information:



This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2803055, IEEE Transactions on Software Engineering

A Systematic Review of Interaction in Search-Based Software Engineering

Aurora Ramírez, José Raúl Romero, *Member, IEEE*, and Christopher L. Simons

Abstract—Search-Based Software Engineering (SBSE) has been successfully applied to automate a wide range of software development activities. Nevertheless, in those software engineering problems where human evaluation and preference are crucial, such insights have proved difficult to characterize in search, and solutions might not look natural when that is the expectation. In an attempt to address this, an increasing number of researchers have reported the incorporation of the 'human-in-the-loop' during search and interactive SBSE has attracted significant attention recently. However, reported results are fragmented over different development phases, and a great variety of novel interactive approaches and algorithmic techniques have emerged. To better integrate these results, we have performed a systematic literature review of interactive SBSE. From a total of 669 papers, 26 primary studies were identified. To enable their analysis, we formulated a classification scheme focused on four crucial aspects of interactive search, i.e. the problem formulation, search technique, interactive approach, and the empirical framework. Our intention is that the classification scheme affords a methodological approach for interactive SBSE. Lastly, as well as providing a detailed cross analysis, we identify and discuss some open issues and potential future trends for the research community.

Index Terms—Search-Based Software Engineering, Interaction, Systematic Literature Review, Optimization

1 Introduction

The design and development of complex, large-scale software systems can be non-trivial and challenging for the software engineer to perform. In an attempt to

based problems in the software development lifecycle, this may also be challenging [6] (chapter 9).

In addition, results of automated search approaches

Thanks!

Fran Buontempo

@fbuontempo
frances.buontempo@gmail.com
overload@accu.org
https://pragprog.com/book/fbmach/genetic-algorithms-and-machine-learning-for-programmers

Chris Simons

@chrislsimons
chris.simons@uwe.ac.uk
www.fet.uwe.ac.uk/~clsimons

UWE Bristol | University of the West of England