Simply the Best: Optimising with an Evolutionary Computing Framework

Frances Buontempo

@fbuontempo
frances.buontempo@city.ac.uk
frances.buontempo@gmail.com
overload@accu.org
www.city.ac.uk/people/academics/frances-buontempo/

Department of Computer Science,
City, University of London,
Northampton Square,
London, EC1V 0HB, United Kingdom.

Chris Simons

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

Department of Computer Science and Creative Technologies,
University of the West of England,
Bristol, BS16 1QY, United Kingdom.

ACCU 2018

April 2018

Workshop slides can be found at

http://www.cems.uwe.ac.uk/~clsimons/ACCU2018/SimplyTheBest.pdf

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

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Workshop

Evolutionary computing

Frameworks for evolutionary computing

Java Class Library for Evolutionary Computing

Optimisation problems:

- ‘OneMax’ Problem
- How to program your way out of a paper bag
- Travelling Salesman Problem

Workshop

Slides

Evolutionary computing

Frameworks for evolutionary computing

Fran & Chris

Programming

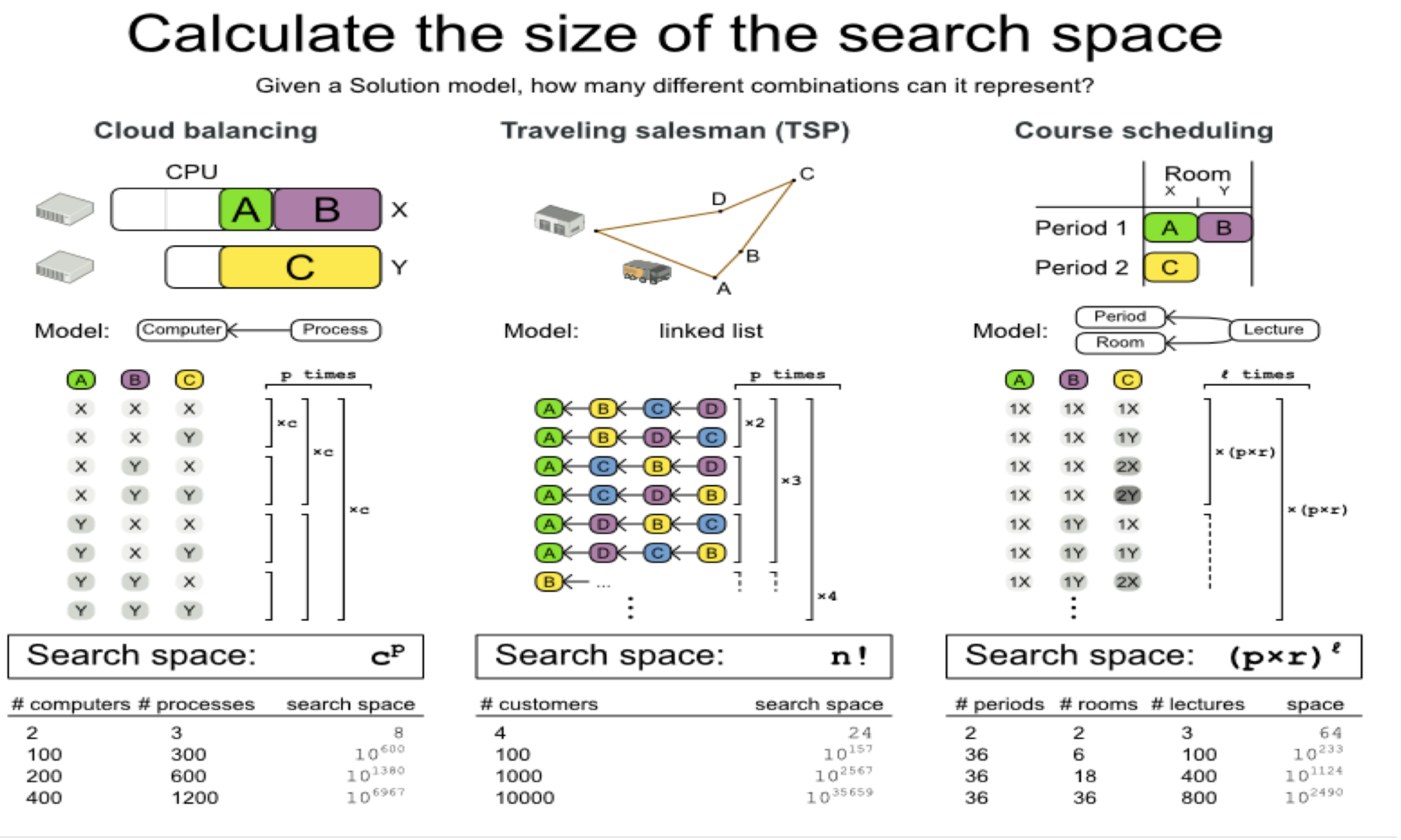
Java Class Library for Evolutionary Computing

Optimisation Problems:

- 'OneMax' Problem
- How to program your way out of a paper bag
- Travelling Salesman Problem

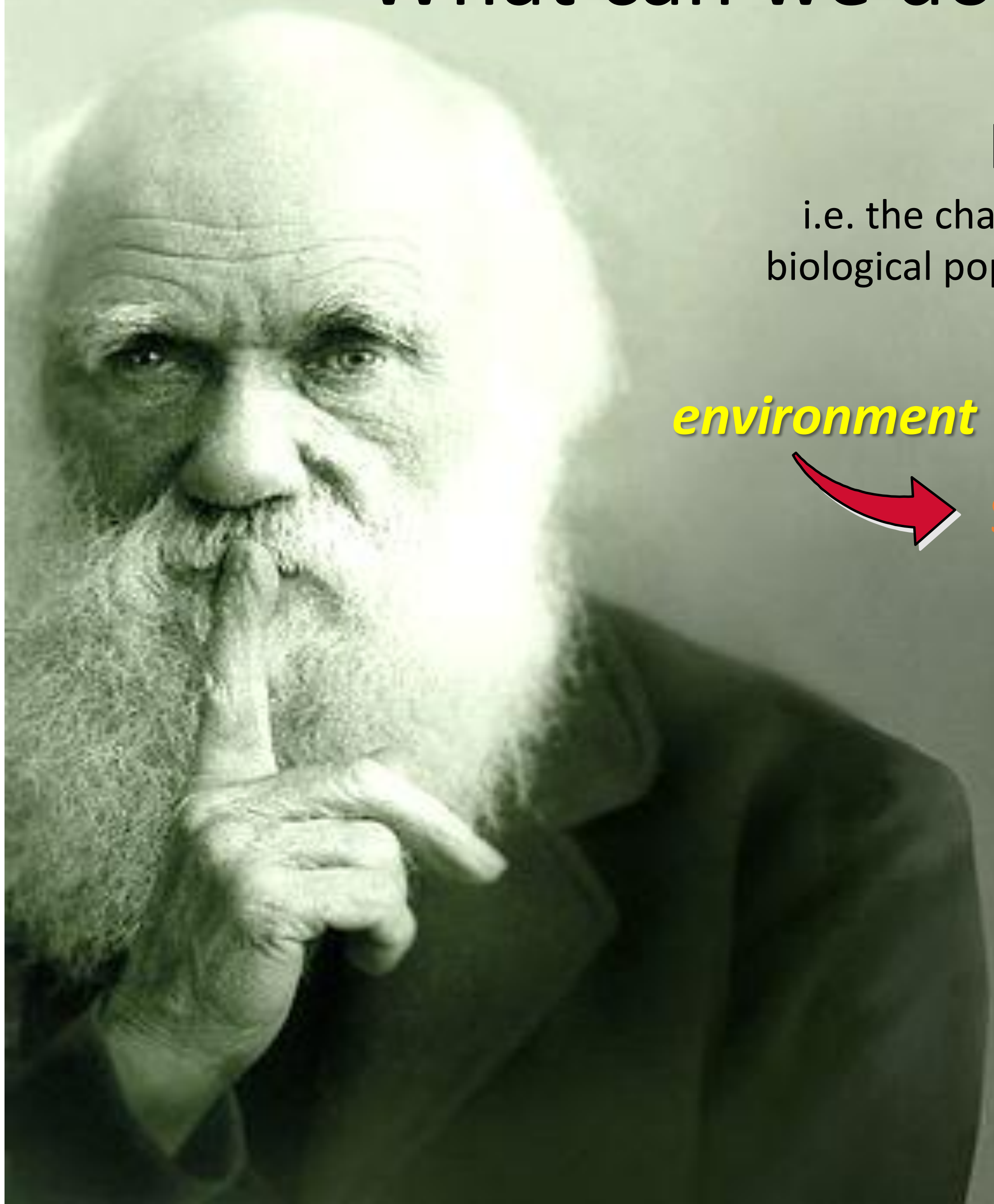
Everyone!

What’s the problem? combinatorial explosion



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

What can we do about it?

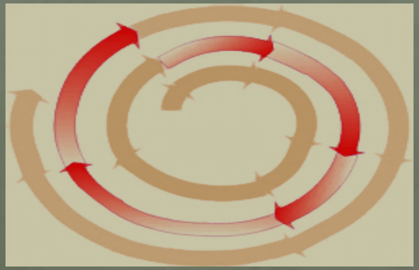


natural evolution

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
```

Eiben, A.E., Smith, J.E. (2015) *Introduction to Evolutionary Computing*, 2nd Edition, Springer.

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***.

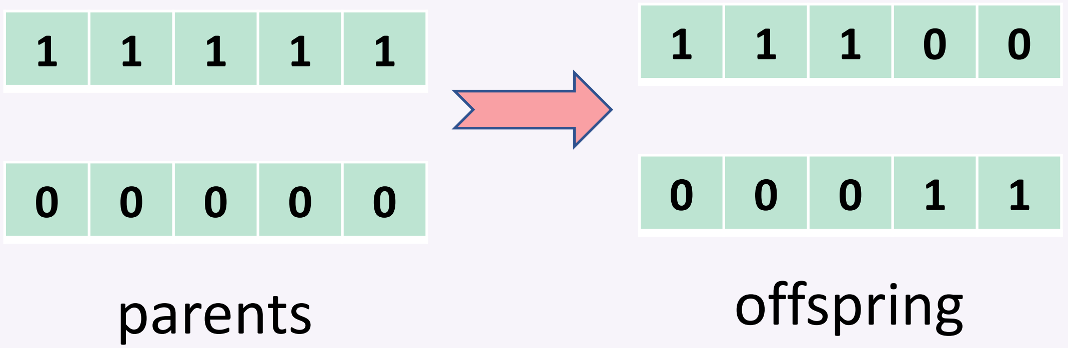
Ideas from biology (2)

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

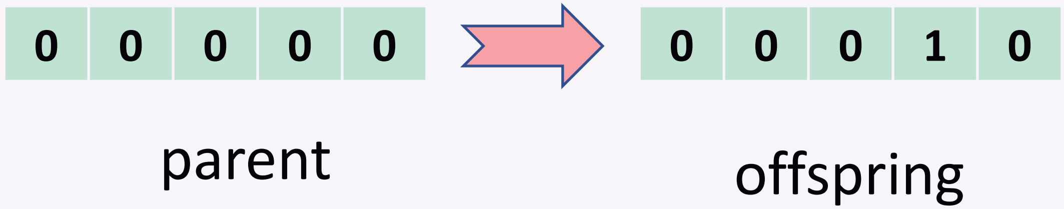
Only the fittest solution ***individuals*** are ***selected*** to breed ***offspring;*** individuals can enter a ***tournament***, the fittest wins the right to breed.

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

Frameworks for Evolutionary Computing

Characteristics include:

- adaptable search components to create customised implementations;
- 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;
- general utilities to conduct experiments, including batch processing , parallel execution; and
- designed with ***best practices*** and ***design patterns*** in mind.

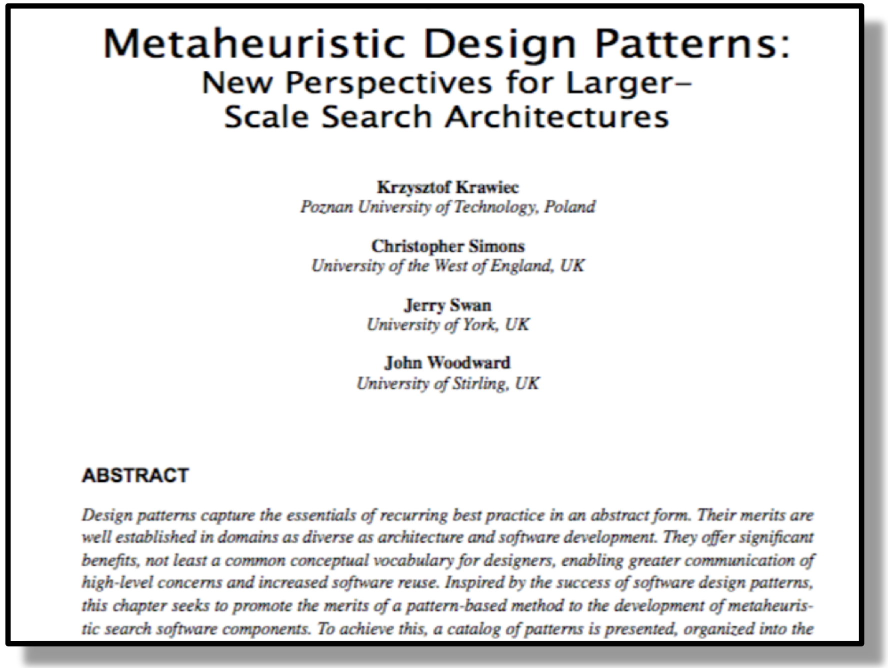
Language	Framework	version	Date
C++	Evolutionary Computation Framework (ECF)	1.4.2	2017
	Evolving Objects (EO)	1.3.1	2012
	jMetalCpp	1.7	2016
	Mallba	2.0	2009
	Open Beagle	3.0.3	2007
	OptFrame	2.2	2017
	PaGMO	2.5	2017
	ParadisEO	2.0.1	2012
Java	Java-based Evolutionary Computation Research System (ECJ)	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/

A pattern-based approach to optimisation



Metaheuristic Design Patterns:
New Perspectives for Larger–Scale Search Architectures

Krzysztof Krawiec
Poznan University of Technology, Poland

Christopher Simons
University of the West of England, UK

Jerry Swan
University of York, UK

John Woodward
University of Stirling, UK

ABSTRACT

Design patterns capture the essentials of recurring best practice in an abstract form. Their merits are well established in domains as diverse as architecture and software development. They offer significant benefits, not least a common conceptual vocabulary for designers, enabling greater communication of high-level concerns and increased software reuse. Inspired by the success of software design patterns, this chapter seeks to promote the merits of a pattern-based method to the development of metaheuristic search software components. To achieve this, a catalog of patterns is presented, organized into the

For a specific problem, need to consider:

Representation

- how to encode a candidate solution…

Fitness

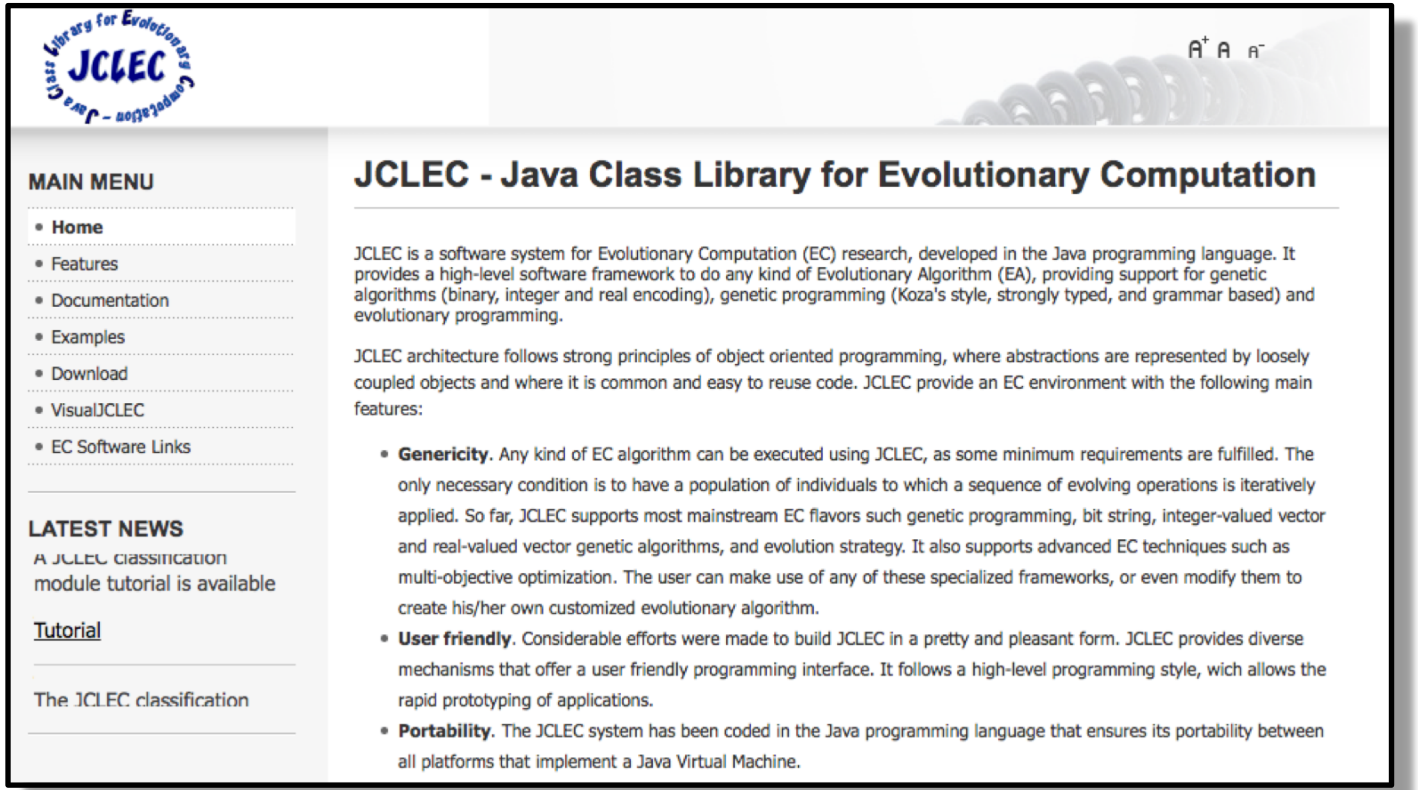
- how to evaluate the fitness of a candidate solution…

Diversity

- how to make offspring different to parents…

Krawiec,, K., Simons, C.L., Swan, J. and Woodward, J. (2017) "Metaheuristic Design Patterns: New Perspectives for Larger-Scale Architectures", in *Handbook of Research on Emergent Applications of Optimization Algorithms,* Vasant, Alparslan-Gok, Weber, Eds., IGI Global Publishing, pp.1-36. DOI: [10.4018/978-1-5225-2990-3.ch001](10.4018/978-1-5225-2990-3.ch001)

Time to look at an example of a evolutionary computing framework


A+ A A-

MAIN MENU

- Home
- Features
- Documentation
- Examples
- Download
- VisualJCLEC
- EC Software Links

LATEST NEWS

A JCLEC classification module tutorial is available

Tutorial

The JCLEC classification

JCLEC - Java Class Library for Evolutionary Computation

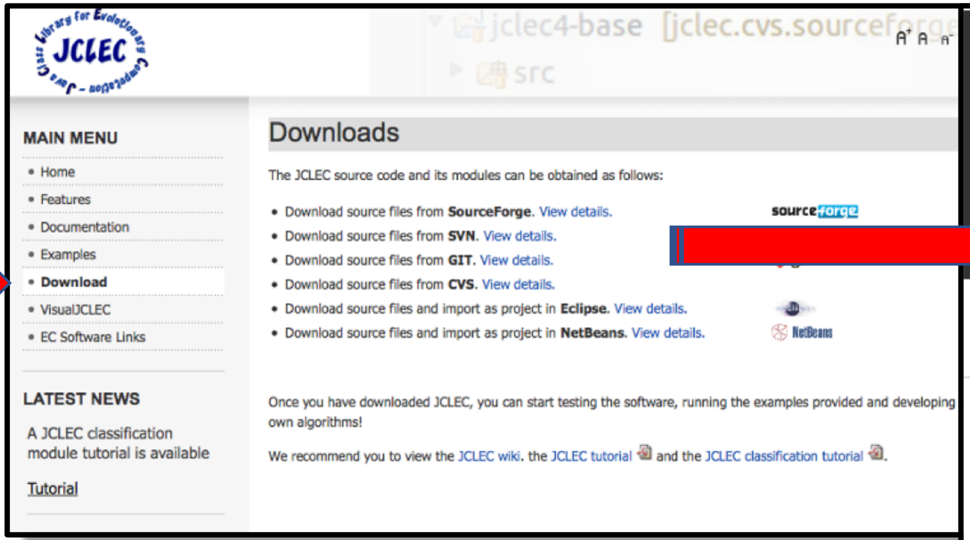
JCLEC is a software system for Evolutionary Computation (EC) research, developed in the Java programming language. It provides a high-level software framework to do any kind of Evolutionary Algorithm (EA), providing support for genetic algorithms (binary, integer and real encoding), genetic programming (Koza's style, strongly typed, and grammar based) and evolutionary programming.

JCLEC architecture follows strong principles of object oriented programming, where abstractions are represented by loosely coupled objects and where it is common and easy to reuse code. JCLEC provide an EC environment with the following main features:

- **Genericity**. Any kind of EC algorithm can be executed using JCLEC, as some minimum requirements are fulfilled. The only necessary condition is to have a population of individuals to which a sequence of evolving operations is iteratively applied. So far, JCLEC supports most mainstream EC flavors such genetic programming, bit string, integer-valued vector and real-valued vector genetic algorithms, and evolution strategy. It also supports advanced EC techniques such as multi-objective optimization. The user can make use of any of these specialized frameworks, or even modify them to create his/her own customized evolutionary algorithm.
- **User friendly**. Considerable efforts were made to build JCLEC in a pretty and pleasant form. JCLEC provides diverse mechanisms that offer a user friendly programming interface. It follows a high-level programming style, wich allows the rapid prototyping of applications.
- **Portability**. The JCLEC system has been coded in the Java programming language that ensures its portability between all platforms that implement a Java Virtual Machine.

http://jclec.sourceforge.net

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



We need this

But we don't need this

And we need this

Or if you prefer…

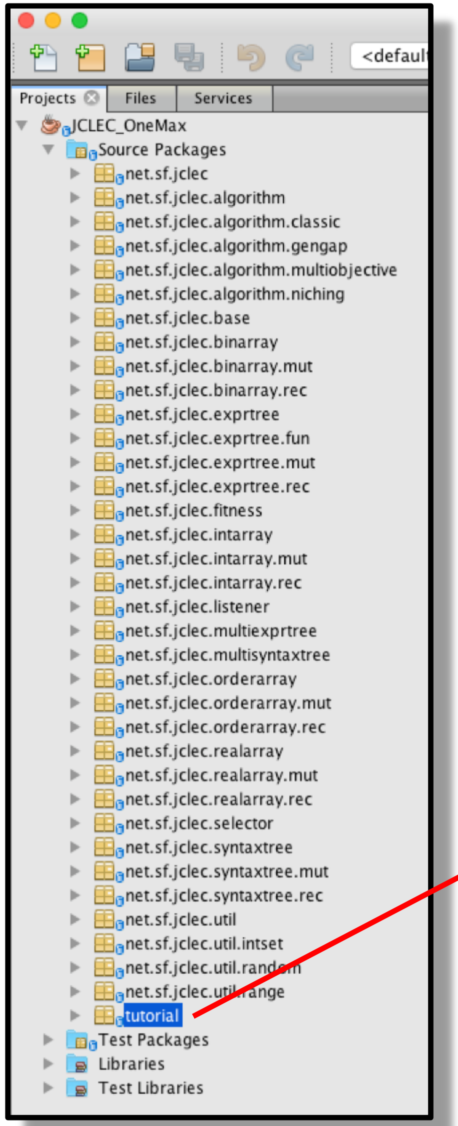
Clone just the files required for the workshop from a github repo:

https://github.com/doctorlove/simplythebest

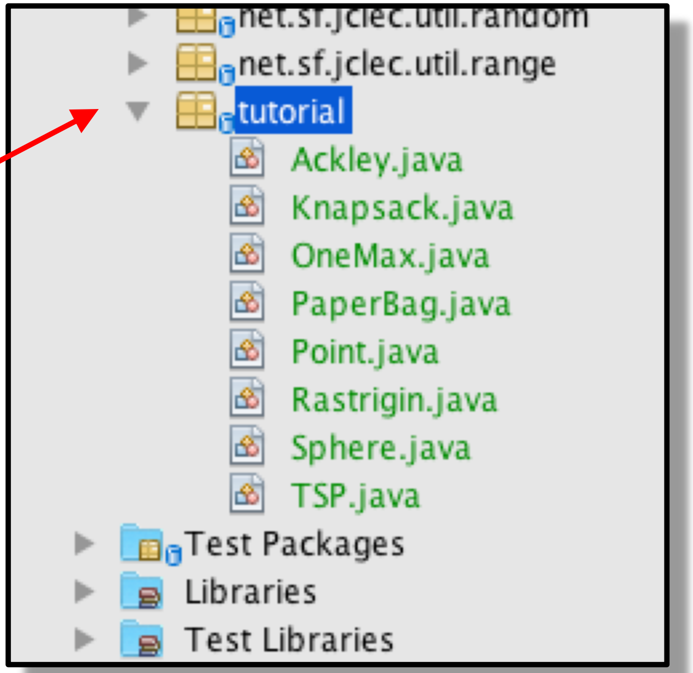
Clone workshop files from a github repo with dependencies managed by Maven:

https://github.com/richriley/jclec-tutorial

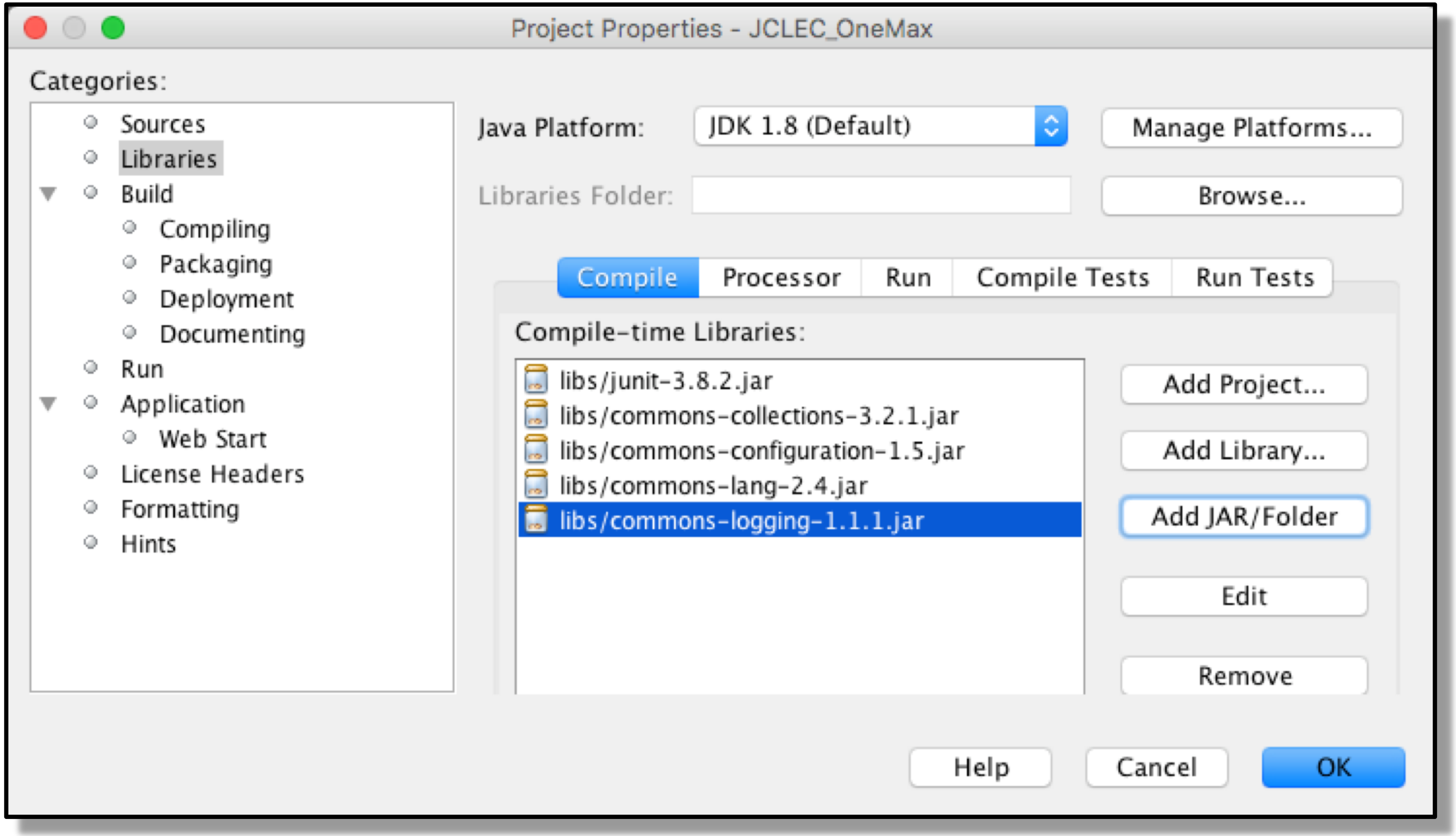
Or copy extracted JCLEC source files to an IDE of your choosing, e.g.



and place tutorial source files in a package called ‘tutorial’



Let the IDE know about the required libraries, i.e.

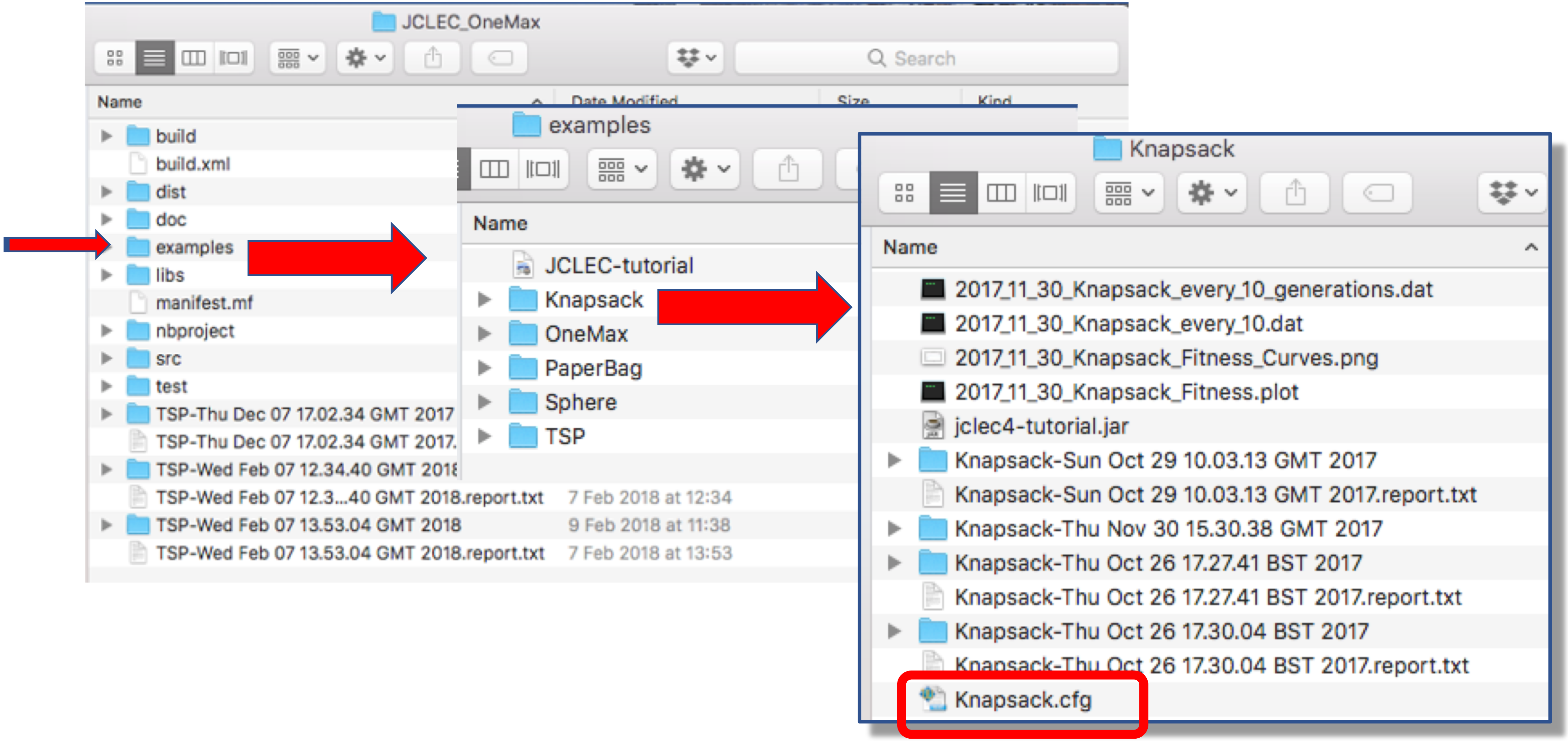


Each JCLEC project has an XML configuration file to specify:

1. Evolutionary algorithm components used, and
2. parameter set up

XML configuration files have a `.cfg` suffix

Copy tutorial examples to a folder called 'examples', e.g.



Let's look at the Knapsack Problem as an example

Given a set of items, each with a weight and a value, the objective is to determine the number of each item to include in the knapsack, so that the total weight is less than a given limit and the total value is as large as possible (i.e. maximised).

Only whole objects can be put in the knapsack if there is enough space for them, partial objects are not allowed.

To achieve a solution, for each object we must decide whether it is placed in the knapsack or not. Let's assume that each object has a variable associated with it that takes the value 1 if the object is placed in the knapsack, or 0 otherwise.

Some problem specifics

OBJECT	WEIGHT	VALUE
1	150	20
2	325	40
3	600	50
4	805	36
5	430	25
6	1200	64
7	770	54
8	60	18
9	930	46
10	353	28

Maximum weight of knapsack is 4200 grams.

There are ten types of object, and the number of article per type cannot exceed ten.

Let’s apply some patterns

1st	1	0	1	1	0	0	0	0	0	0
2nd	1	0	1	1	0	0	1	0	0	0
3rd	1	0	1	1	0	0	0	0	1	0
…	…									
10th	1	0	1	1	0	0	0	1	0	0

Representation

- how to encode a candidate solution?

Array of 100 bits

Fitness

- how to evaluate the fitness of a candidate solution?

Summation of value of objects in Knapsack

(Infeasible individuals assigned a negative value)

Diversity

- how to make offspring different to parents?

Recombination (i.e. two parents => two offspring)

Mutation (i.e. one parent => one offspring)

Now let's select the evolutionary algorithm…

Initialisation: each individual solution in the population is randomly initialized.

Evolution: simple generational approach with elitism (SGE),

i.e. all offspring go forward to the next generation, and

the best individual of this generation is included in the next generation of the algorithm.

Replacement strategy: the best offspring automatically replaces the worst parent individual.

… and some parameters

Population size: 100 individuals.

Stop Criterion: 1000 generations.

Parent selection: tournament between 2 individuals selected at random from the population.

The configuration file, ‘knapsack.cfg’ (1)

```
<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>100</max-of-generations>
  <species type="net.sf.jclec.binarray.BinArrayIndividualSpecies”
     genotype-length="100"/>
```

Continued…

The configuration file, ‘knapsack.cfg’ (2)

```
<evaluator type="tutorial.Knapsack">
  <products>
    <max-each-product>10</max-each-product>
    <product weight="150" profit="20"/>
    <product weight="325" profit="40"/>
    <product weight="600" profit="50"/>
    <product weight="805" profit="36"/>
    <product weight="430" profit="25"/>
    <product weight="1200" profit="64"/>
    <product weight="770" profit="54"/>
    <product weight="60" profit="18"/>
    <product weight="930" profit="46"/>
    <product weight="353" profit="28"/>
    <max-weight>4200</max-weight>
  </products>
</evaluator>
```

Continued…

The configuration file, ‘knapsack.cfg’ (3)

```
    <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>10</report-frequency>
      <report-on-file>true</report-on-file>
      <save-complete-population>true</save-complete-population>
      <report-title>Knapsack-</report-title>
    </listener>
  </process>
</experiment>
```

Implementation of fitness, configuration in ‘Knapsack.java’

```
public class Knapsack extends AbstractEvaluator implements IConfigure
{
    List<Integer> weights = new ArrayList<Integer>();
    List<Integer> profits = new ArrayList<Integer>();
    // etc.

    /** Set the maximize flag.
     * @param maximize Actual maximize flag.
     */
    public void setMaximize(boolean maximize){
       this.maximize = maximize;
    }

    public Comparator<IFitness> getComparator() {
        // Set fitness comparator (if necessary)
        if (COMPARATOR == null)
            COMPARATOR = new ValueFitnessComparator(!maximize);
        return COMPARATOR;
    }
```

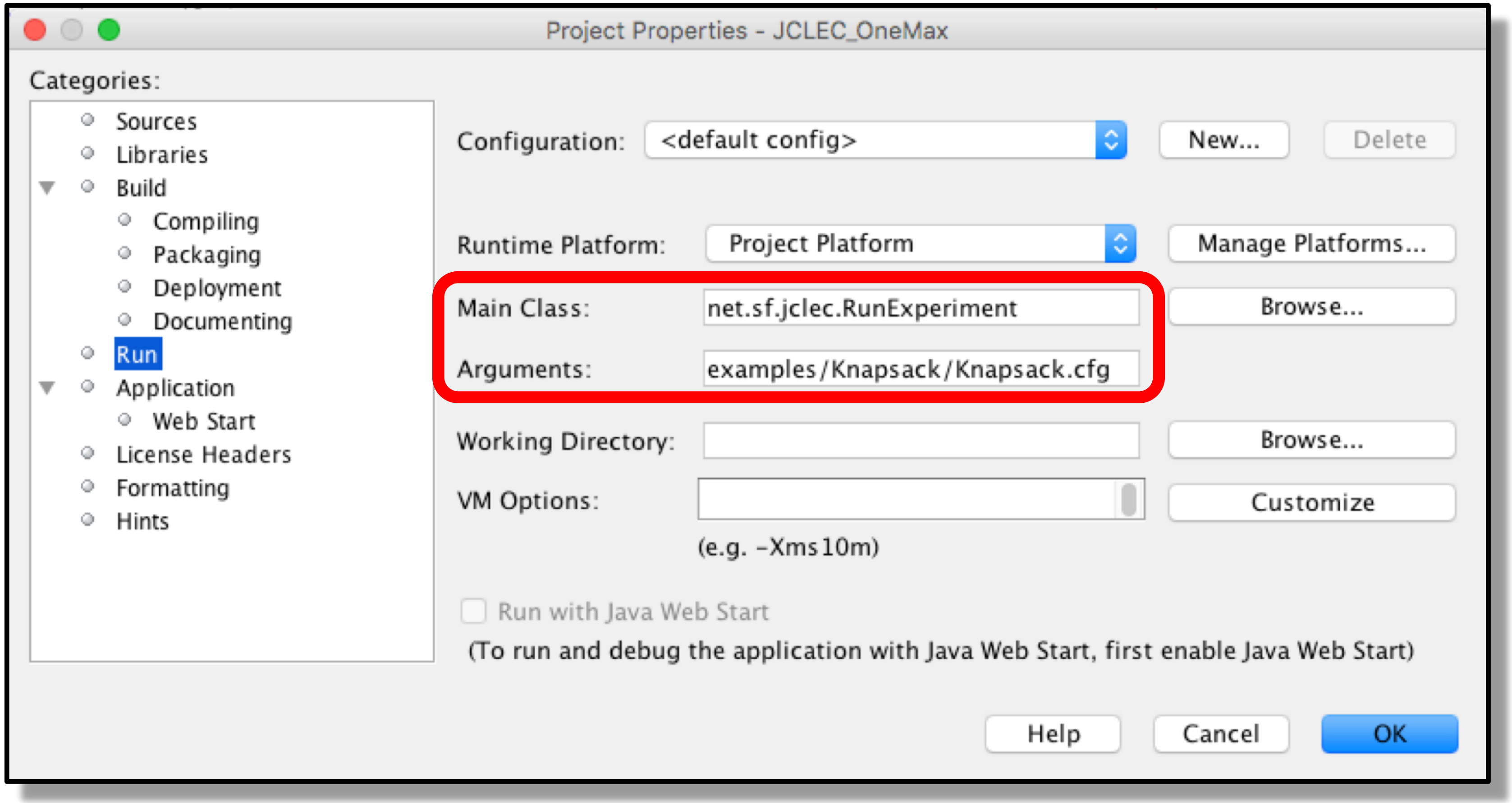
Implementation of configuration

```
public void configure(Configuration settings)
{   //get max-number-products
    maxOfProducts = settings.getInt("products.max-each-product",1);
    // get number-products
    numberOfProducts = settings.getList("products.product[@weight]").size();
    // get max-weight
    maxWeight = settings.getInt("products.max-weight",1);

    for(int i=0; i<numberOfProducts; i++) {
        for(int j=0; j<maxOfProducts; j++) {
            profits.add(settings.getInt("products.product("+i+")[@profit]"));
            weights.add(settings.getInt("products.product("+i+")[@weight]"));
        }
    }
    // Maximize flag
    setMaximize(true);
}
```

Implementation of fitness evaluation

```
@Override
protected void evaluate(IIndividual ind)
{
    byte [] genotype = ((BinArrayIndividual)ind).getGenotype();
    int totalweigth = 0, totalprofit = 0;

    for (int i=0; i<genotype.length; i++) {// Calculate weight, profit
        totalweigth += genotype[i]*weights.get(i);
        totalprofit += genotype[i]*profits.get(i);
    }

    if (totalweigth <= maxWeight){
        ind.setFitness(new SimpleValueFitness(totalprofit));
    }
    else {
        ind.setFitness(new SimpleValueFitness(-totalprofit));
    }
}
```

Set the main class as 'net.sf.jclec.RunExperiment',
and set 'knapsack.cfg' (with path) as an argument to the executable:

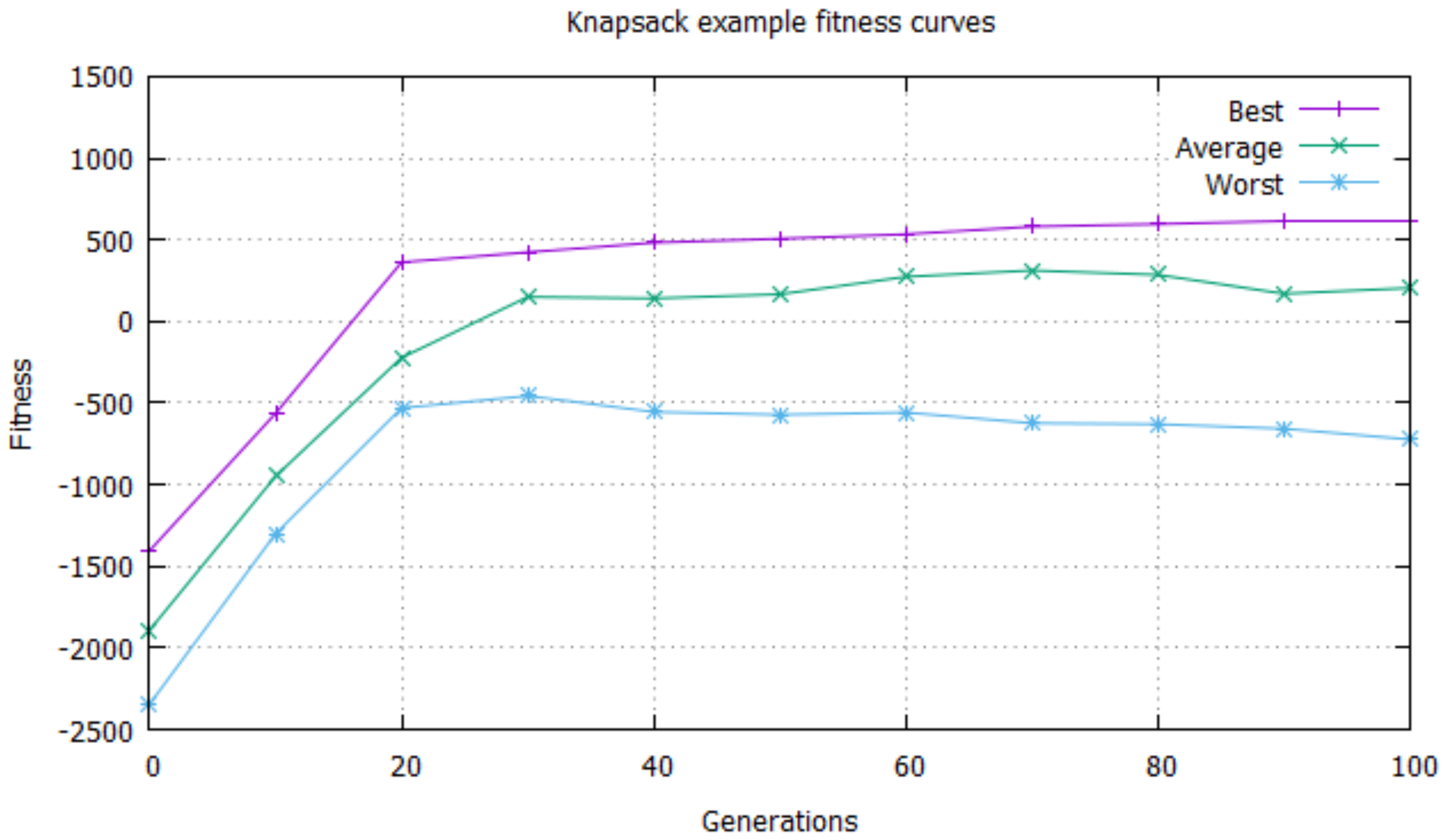


And now we can build and run!

Demonstration

Population fitness curves

Using gnuPlot…



Optimisation Problem (1)

OneMax Problem

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?

?

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)?

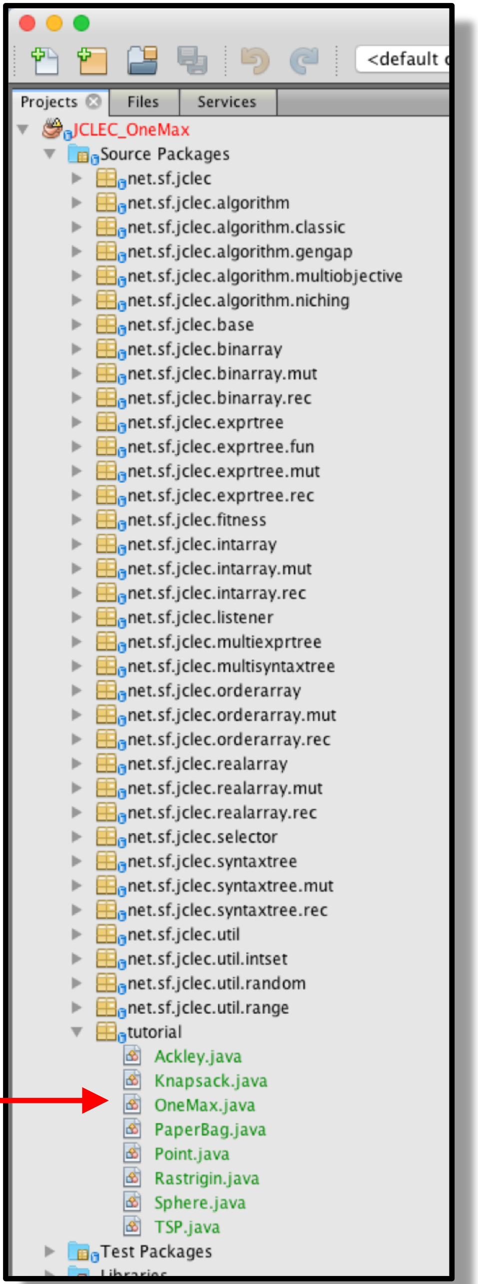
… and suggested parameters

Population size: 100 individuals

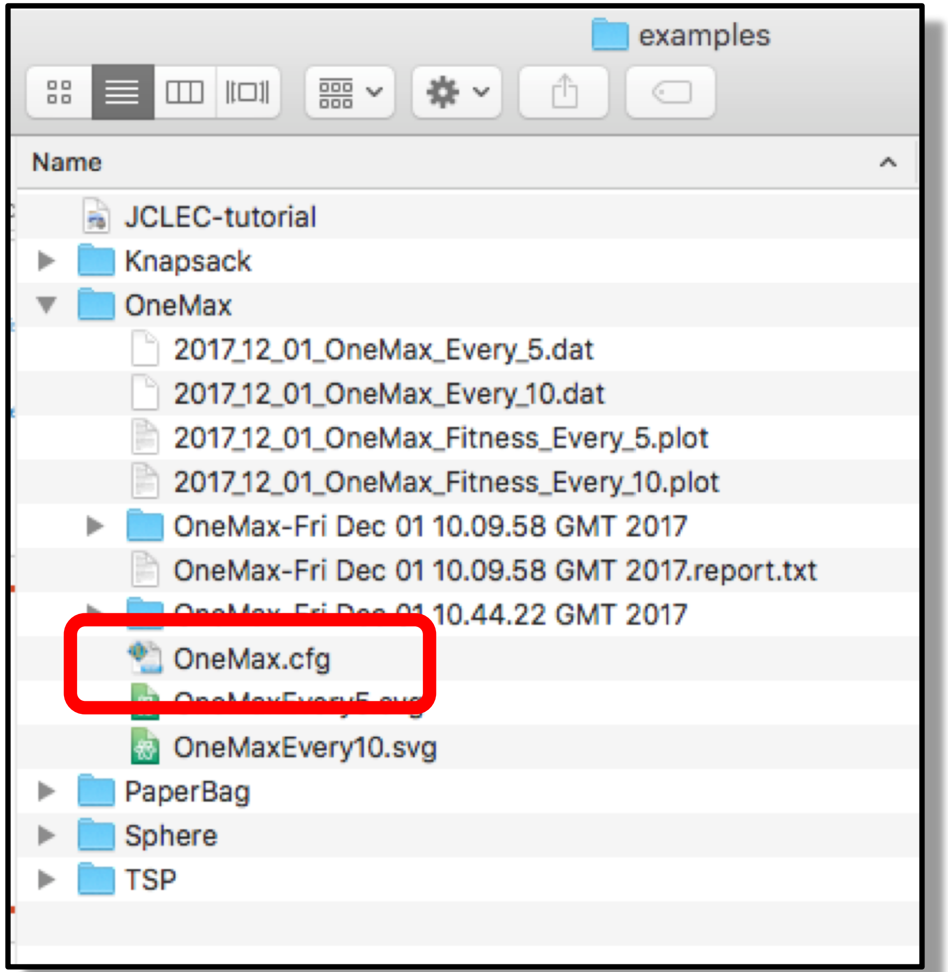
Stop Criterion: 50 generations

Parent selection: tournament of 2 individuals

Make a new file “OneMax.java”



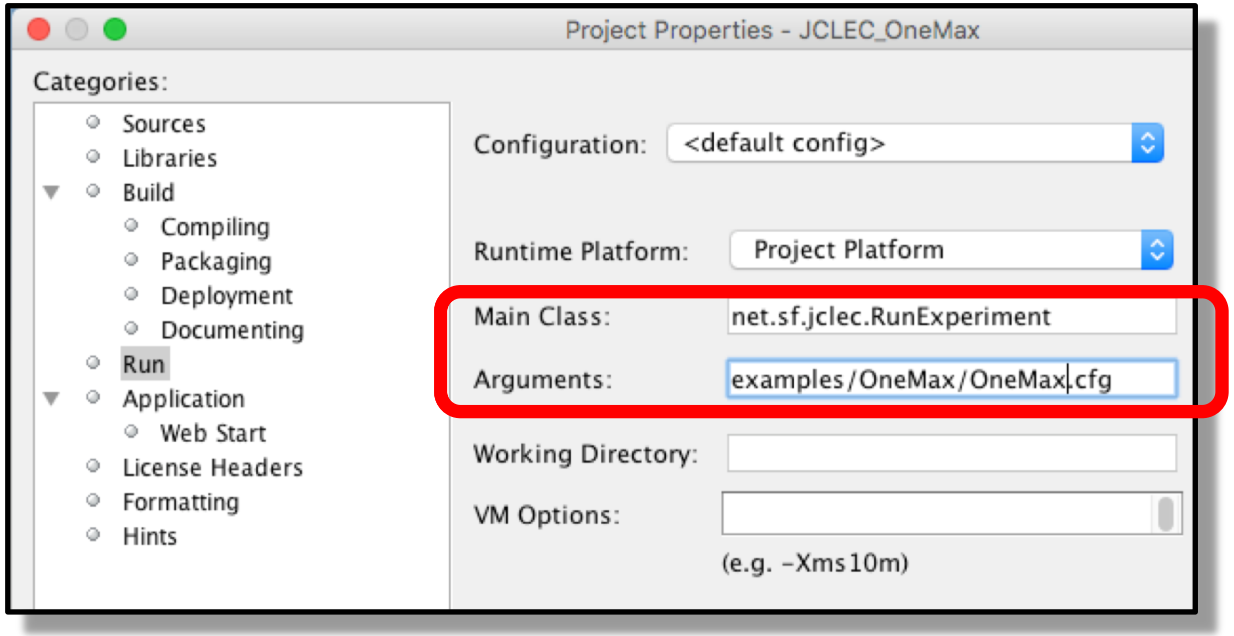
And make a new folder “OneMax” in ”examples” for the configuration file – “OneMax.cfg”



Take inspiration (i.e. copy & edit) from the Knapsack Problem, and enjoy the programming!

Hint: there's no need to program any configuration, other than setting the maximisation flag true.

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; changes to Knapsack.cfg highlighted in **bold**:

```
<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 ) );
}
```

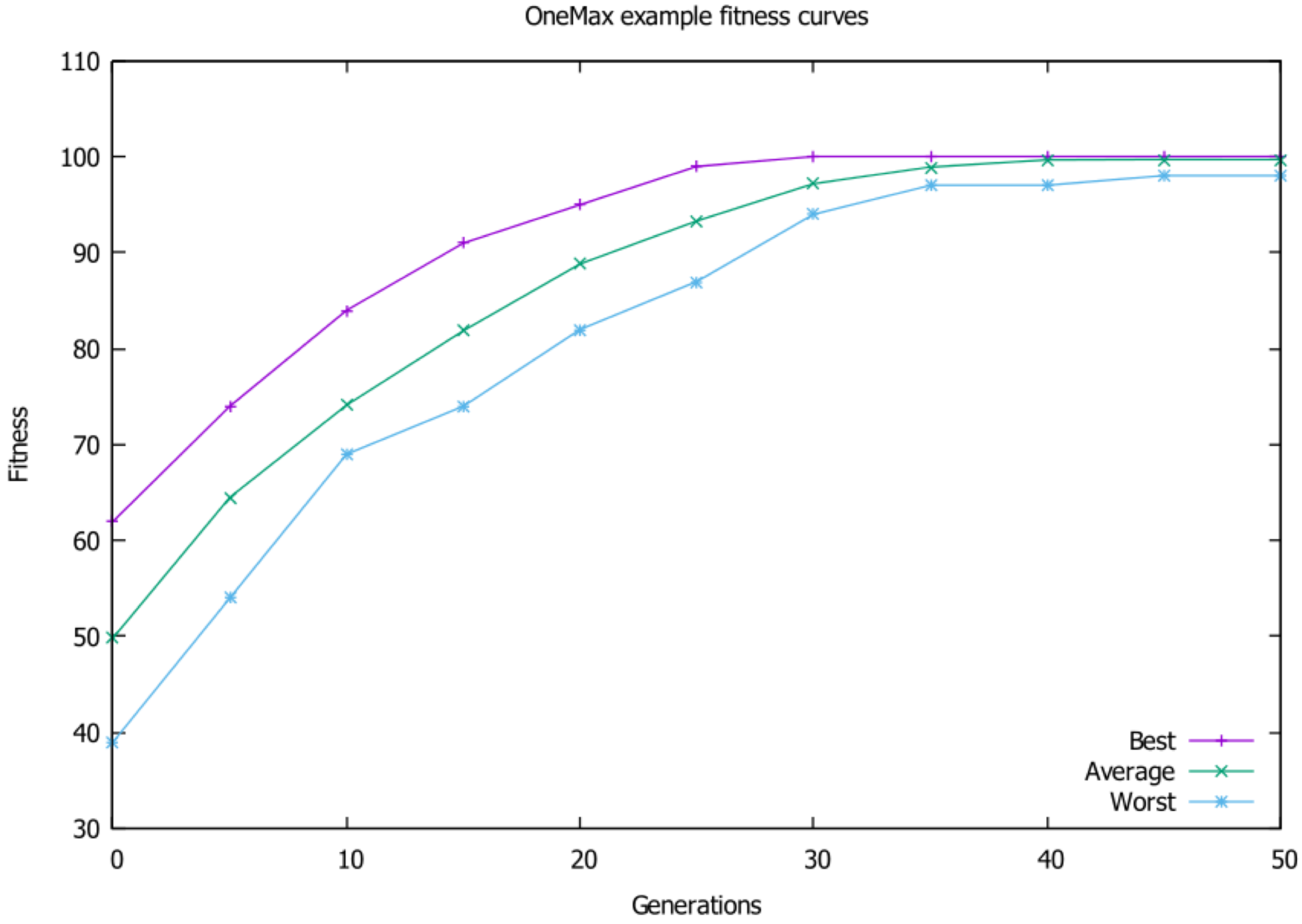
Demonstration

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 = 0.3375999999998929

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)
```

...

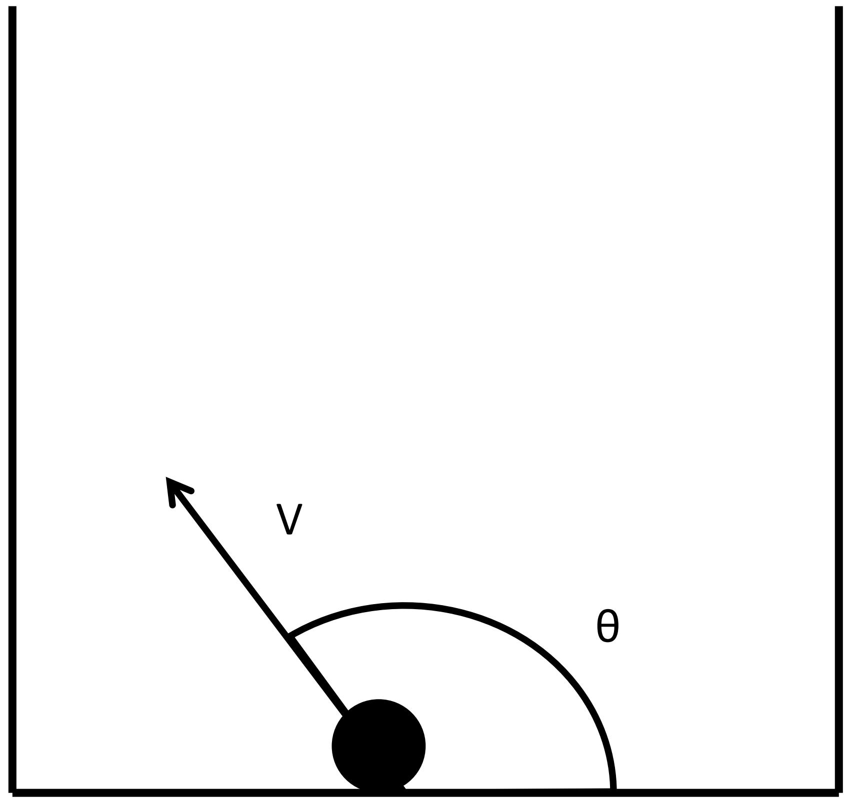
```
,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@45dd4eda[value=99.0]]
1,1,0,1,1,1,1,1,1,0,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@222114ba[value=87.0]]
,1,1,1,1,1,1,1,1,1,0,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@358c99f5[value=94.0]]

,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@41e36e46[value=100.0]]
1,1,0,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@229d10bd[value=94.0]]
,1,1,1,1,1,1,1,1,1,1,1},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]]
1,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@67d48005[value=97.0]]
,1,1,1,1,1,1,1,1,1,1,1},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]]
1,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@2e377400[value=97.0]]
,1,1,1,1,1,1,1,1,1,1,1},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]]
1,1,0,0,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@79da8dc5[value=98.0]]
,1,1,1,1,1,1,1,1,1,1,1},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]]
1,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@3e27aa33[value=98.0]]
,1,1,1,1,1,1,1,1,1,1,1},fitness=net.sf.jclec.fitness.SimpleValueFitness@4b29d1d2[value=100.0]]
```


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

Optimisation Problem (2)

Program your way out of a paper bag

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

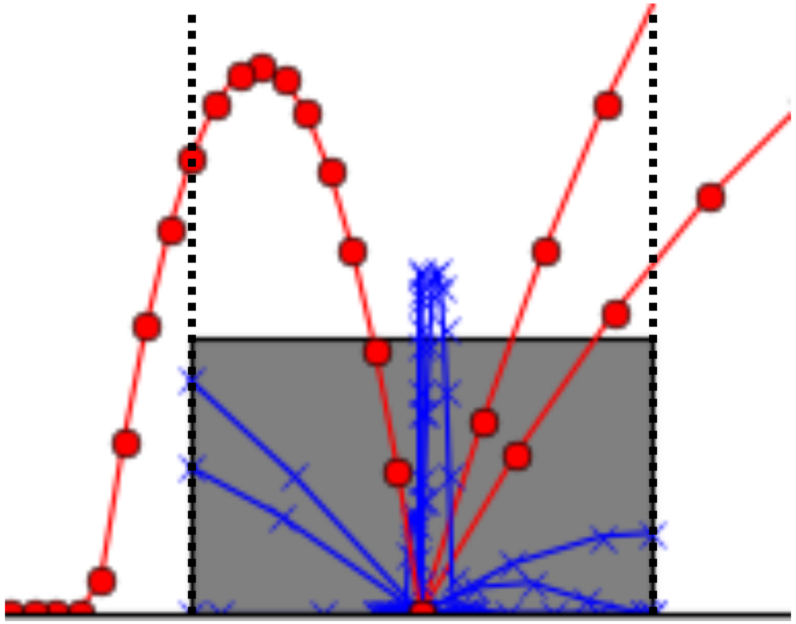
$$x = k + \frac{1}{2}w + vt\cos\theta$$
$$y = vt\sin\theta - \frac{1}{2}gt^2$$

x is the horizontal displacement and ***y*** the vertical displacement. The projectile will just escape when $y \geq h$ 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

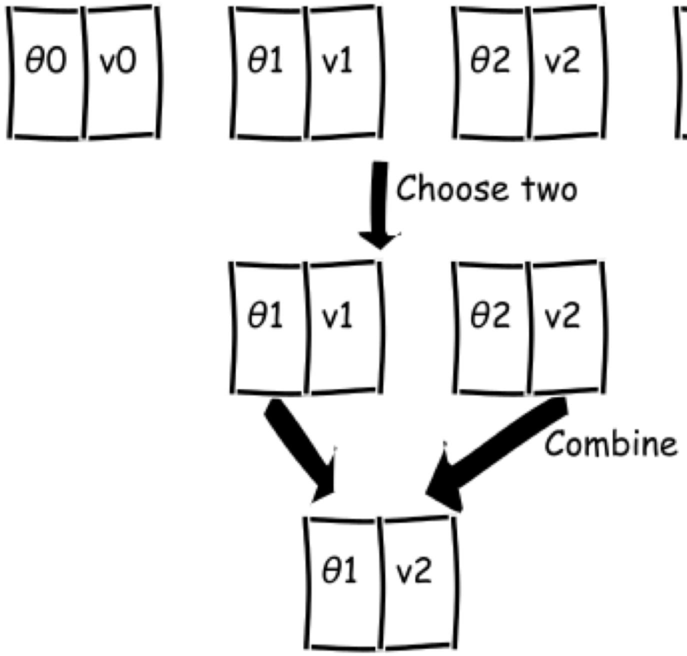
- Launching at random, something either ends in or out of the bag
- But some fail cases are less bad than others



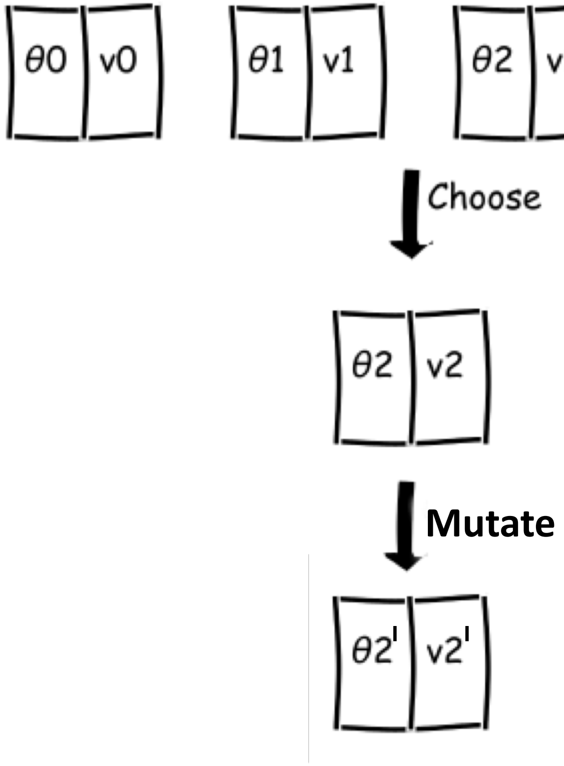
- 3 escape
- 2 on left get “close”
- Could “close” mean height (at edge of bag)?
  - Fitness = height

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

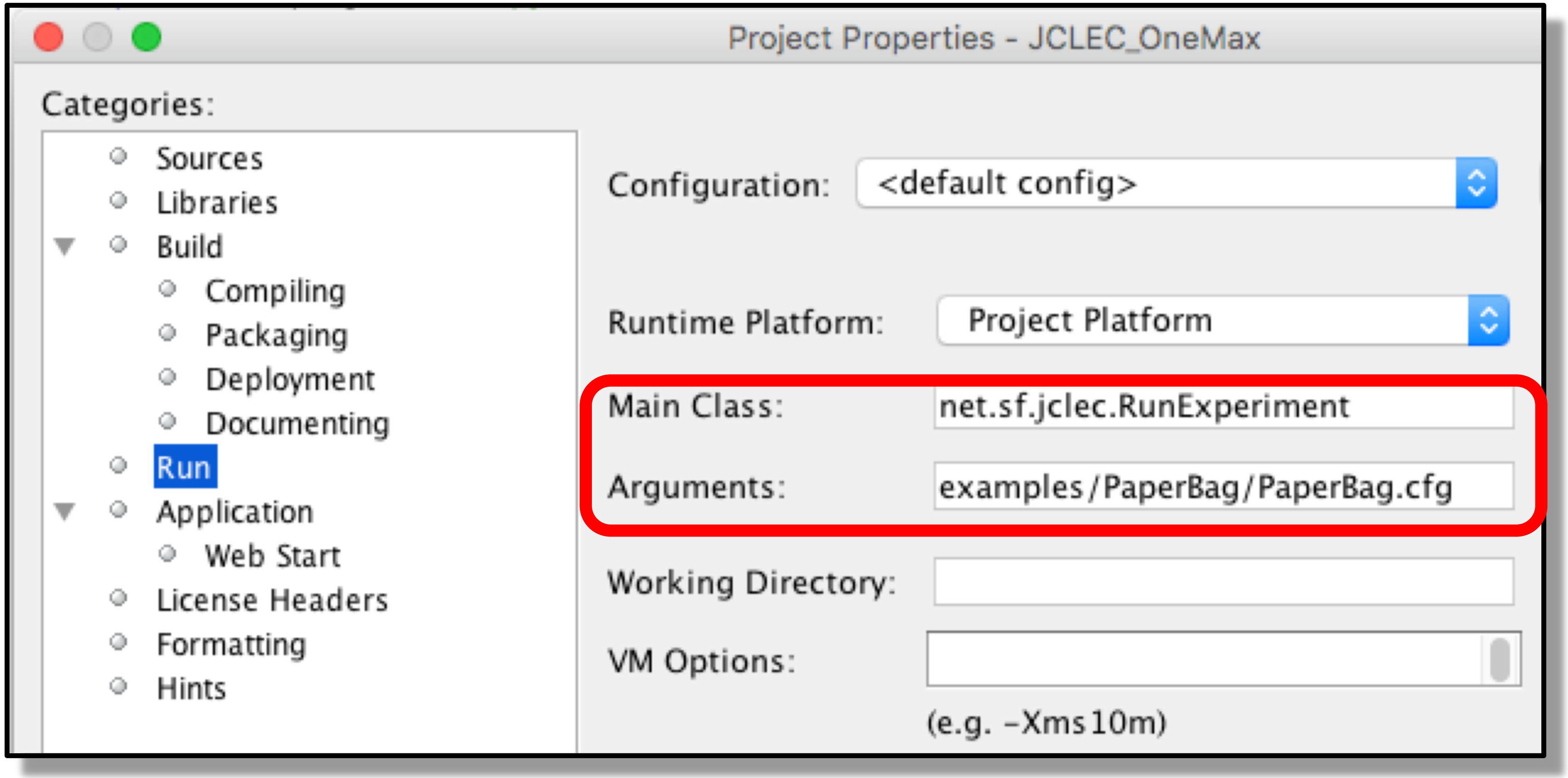
Population size: 12 individuals

Stop Criterion: 20 generations

Parent selection: tournament of 2 individuals

```xml
<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 );
    }

    // …
```

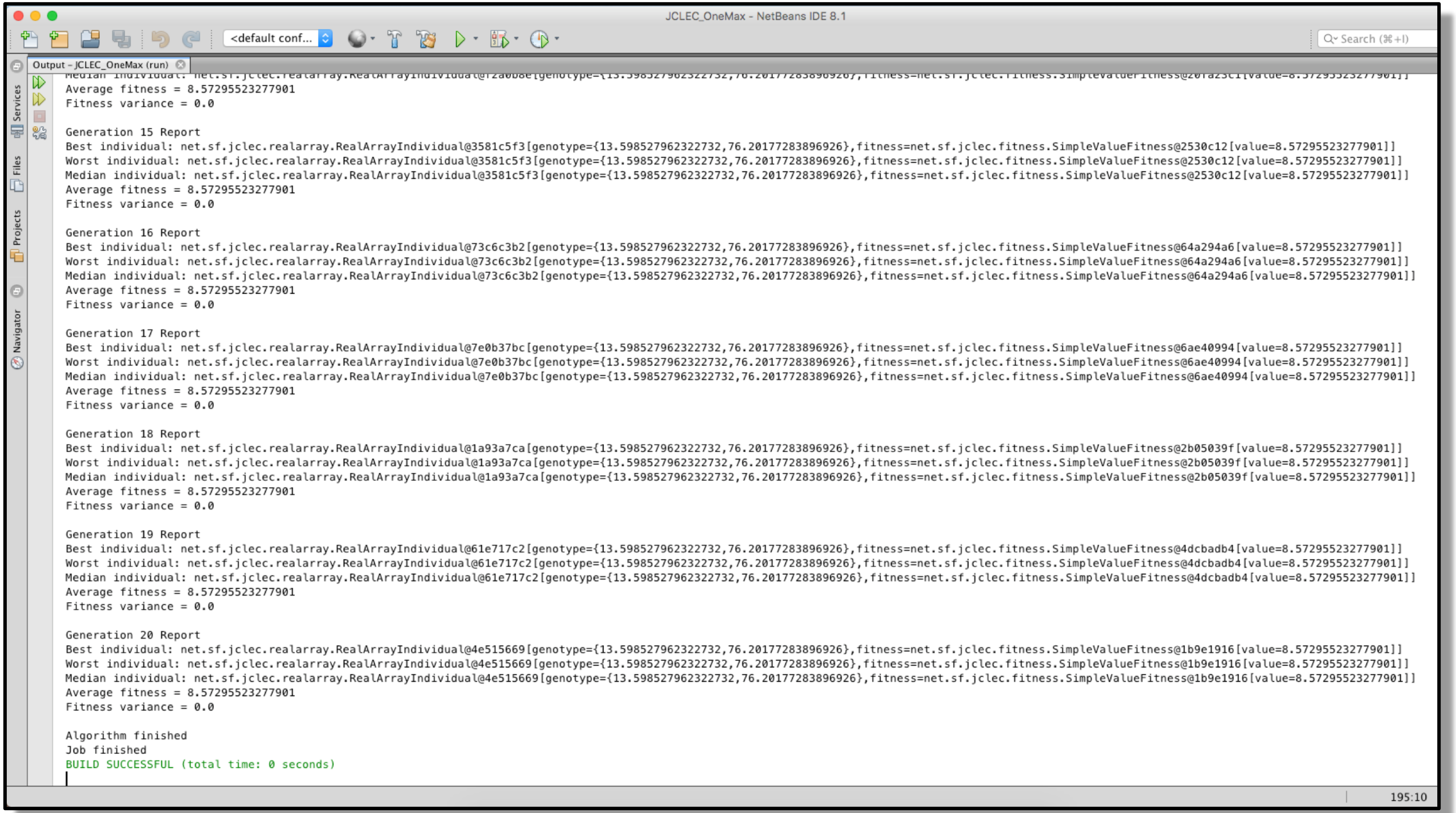
Continued….

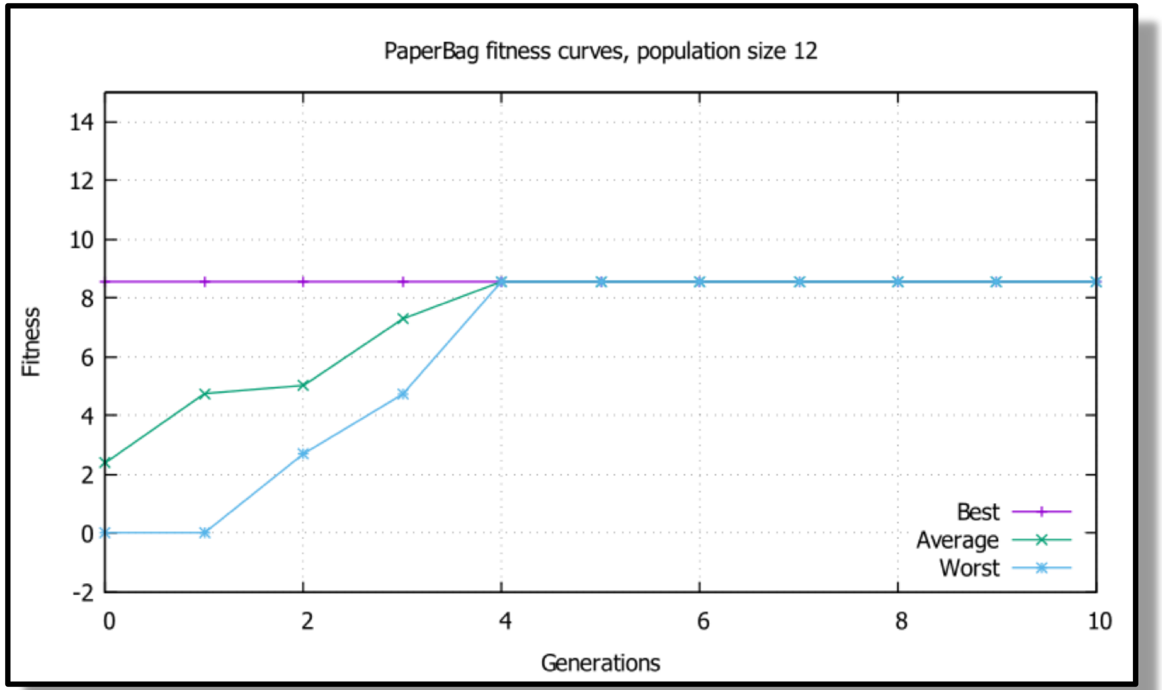
```
        double fitness = 0.0;

        // 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 ) );
    }
```

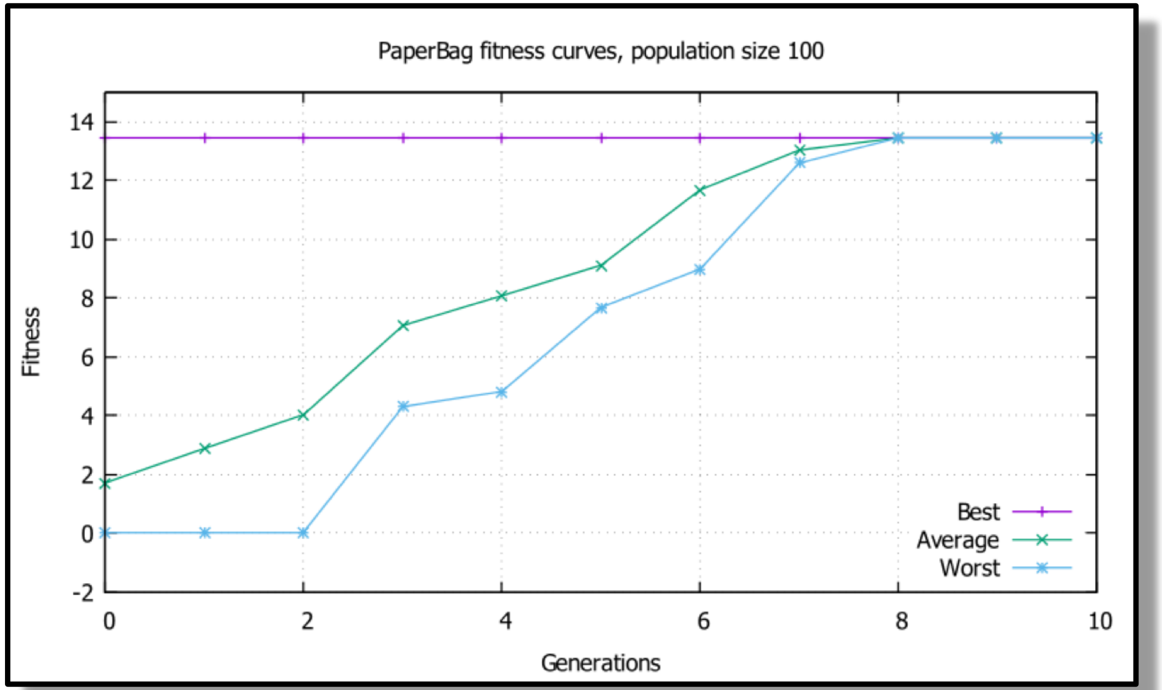
The getPointAtTime( ) method:

```
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

Optimisation Problem (3)

Travelling Salesman Problem

The **travelling salesman problem** (**TSP**) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city (once only)* and returns to the origin city?"

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

* Our clarification

Some problem specifics. Let’s take 52 locations in Berlin:

```
NAME: berlin52
TYPE: TSP
COMMENT: 52 locations in Berlin(Groetschel)
DIMENSION: 52
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 565.0 575.0
2 25.0 185.0
3 345.0 750.0
4 945.0 685.0
...
51 1340.0 725.0
52 1740.0 245.0
EOF
```

‘Berlin.tsp’ files available from the examples on JCLEC

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

Representation

- how to encode a candidate solution in the population?

array of 52 integers,
whose values represent specific cities

Fitness

- how to evaluate the fitness of a candidate solution?

distance of the route
(to be minimised)

Diversity

- how to make offspring different to parents?

Recombination and mutation

Initialisation: random

Evolution: simple generational with elitism (SGE)

… and suggested parameters

Population size: 100 individuals

Stop Criterion: 100 generations

Parent selection: tournament of 2 individuals

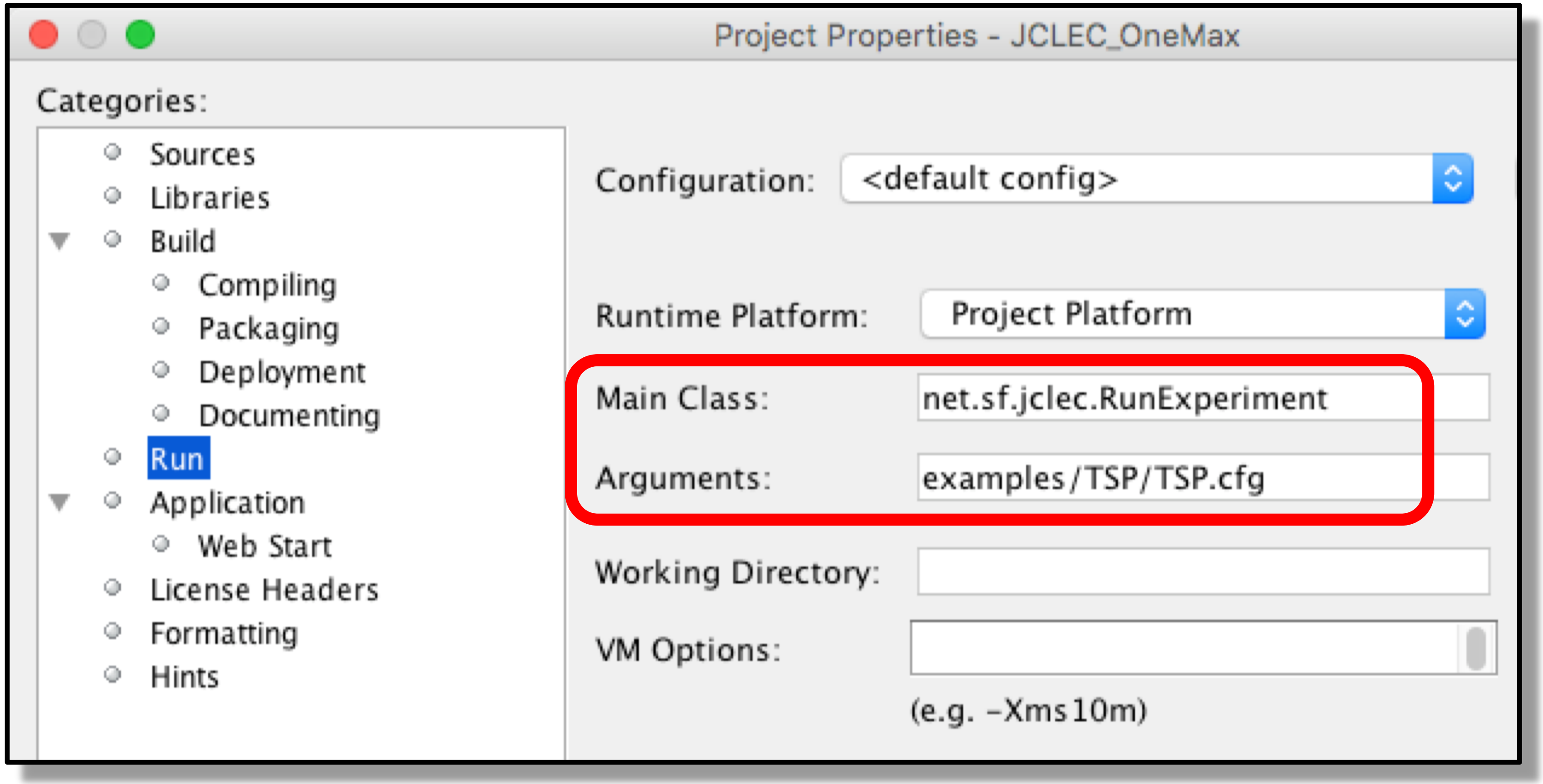
```xml
<experiment>
  <process algorithm-type="net.sf.jclec.algorithm.classic.SGE">
  <rand-gen-factory type="net.sf.jclec.util.random.RanecuFactory" seed="987328938"/>
  <population-size>100</population-size>
  <max-of-generations>1000</max-of-generations>
  <species type="net.sf.jclec.orderarray.OrderArrayIndividualSpecies"
    genotype-length="52"/>
  <evaluator type="tutorial.TSP" file-name="examples/TSP/cities/berlin52.tsp”
    number-cities="52"/>
  <provider type="net.sf.jclec.orderarray.OrderArrayCreator"/>
  <parents-selector type="net.sf.jclec.selector.TournamentSelector”>
  <tournament-size>2</tournament-size>
  </parents-selector>
    <mutator type="net.sf.jclec.orderarray.mut.Order2OptMutator" mut-prob="0.2”/>
    <recombinator type="net.sf.jclec.orderarray.rec.OrderPMXCrossover”
    rec-prob="0.9"/>
  <listener type="net.sf.jclec.listener.PopulationReporter">
    <report-frequency>50</report-frequency>
    <report-on-file>true</report-on-file>
  </listener>
  </process>
</experiment>
```

Here’s one way to solve the TSP with the framework:

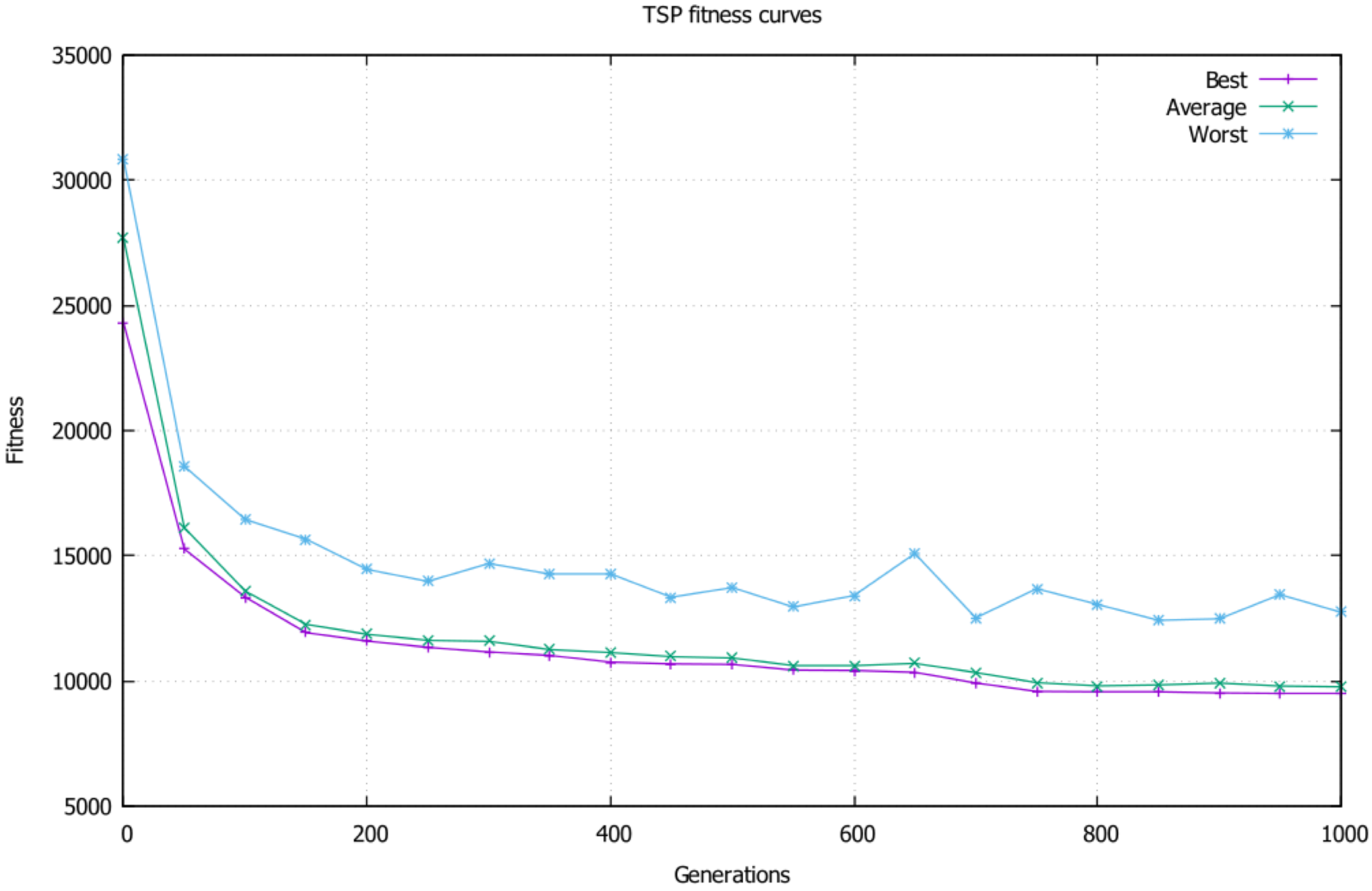
```
public class TSP extends AbstractEvaluator implements IConfigure
{

    /** Maximize of minimize functions? */
    protected boolean maximize = false;

    /** Distances matrix */
    private double distances[][];

    private Comparator<IFitness> COMPARATOR;

    private BufferedReader br;

    // configuration set up etc...
```

```java
protected void evaluate(IIndividual ind){

    // Individual genotype
    int [] genotype = ((OrderArrayIndividual)ind).getGenotype();

    double distance = 0;

    for (int i=0; i<genotype.length-1; i++) {
        distance += distances[genotype[i]][genotype[i+1]];
    }

    distance += distances[genotype[genotype.length-1]][genotype[0]];

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

Invoke the executable with ”TSP.cfg” as an argument



Demonstration


TSP fitness curves
Best
Average
Worst
Fitness
Generations
35000
30000
25000
20000
15000
10000
5000
0
200
400
600
800
1000

Practicalities and final thoughts (1)

We don’t have to ‘re-invent the wheel’ with evolutionary algorithms i.e. we don’t have to start from nothing.

Rather, we can focus programming on
 (i) algorithm configuration, by providing a configuration file
 (and sometimes implementing `IConfigure`)
 (ii) fitness/cost measures, by extending `AbstractEvaluator`

Practicalities and final thoughts (2)

Using an evolutionary computing framework for optimization

- (correctly) focusses attention on
  - (i) Problem characteristics e.g. solution representation, fitness/cost measures, constraints
  - (ii) metaheuristic design patterns and best practices
- can dramatically reduce algorithm implementation time and issues

although

- learning curve investment can lead to framework lock-in?

Acknowledgement

Fran and Chris would like to thank Aurora Ramirez and the research staff at the Knowledge Discovery and Intelligent Systems (KDIS) research group, University of Cordoba, Spain, for advice with JCLEC.

Thanks!

Contact details

Fran Buontempo

@fbuontempo
frances.buontempo@city.ac.uk
frances.buontempo@gmail.com
overload@accu.org
www.city.ac.uk/people/academics/frances-buontempo

Chris Simons

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