Assignment 3
Multiobjective optimisation and a real-world problem

April 11, 2016

1 Is the problem non-trivial?

1.1 Non-trivial

A non-trivial multi-objective optimization problem is a problem for which there does not exist a single solution that simultaneously optimizes each objective. If this is the case then the objective functions are said to be conflicting, and there exists a (possibly infinite) number of Pareto optimal solutions. A solution is called Pareto optimal if none of the objective functions can be improved in value without degrading some of the other objective values, these values make up the Pareto front[1].

The optimal design of a pinned-pinned sandwich beam problem is to choose a set of optimal design (decision) variables x = [d1, d2, d3, b, L] in order to (a) minimise the frequency of the vibratory disturbance; and (b) minimise the total cost of building the beam. As these objectives are conflicting and there is no singe solution that will simultaneously optimize each of these objectives the problem is non-trivial.

2 Implementation

2.1 Design Decisions

2.1.1 MOEA Framework

The MOEA frame work was used for implementation. The MOEA framework is a free and open source Java library for developing and experimenting with multiobjective evolutionary algorithms(MOEAs) and other general-purpose multiobjective optimization algorithms"[2]. This library was utilised as the framework is easy to use and it requires minimum effort in order to define a problem, there is also very good documentation provided.

[1] https://en.wikipedia.org/wiki/Multiobjectiveoptimization
[2] http://moeaframework.org

2.1.2 Algorithms

The two algorithms chosen to solve the multi-optimisation problem were NSGII and GDE3 as these are both popularly used multiobjective evolutionary algorithms.

2.2 Implementation Details

I based my implementation on the example solutions provided, the same class was used for plotting the pareto front and executing the algorithms.

2.2.1 Problem

The problem comprises of 3 core parts, firstly how we represent the properties of the materials within the problem. Secondly how the objectives and constraints are represented and finally how we construct a new solution.

The problem is defined by extending the AbstractProblem class and inheriting the required methods.

```
//create a 3 by 3 array to represent the materials
  private double [][] pEc = new double[3][3];

  public PPSBProblem() {
     super(5,2,3);

     //pEc p=density E=young's constant c=Cost of material
     //properties of material 1
     pEc[0][0] = 100.0;
     pEc[0][1] = 1.60E+09;
     pEc[0][2] = 500.0;
     //properties of material 2
     pEc[1][0] = 2770.0;
     pEc[1][1] = 7.00E+10;
     pEc[1][2] = 1500.0;
     //properties of material 3
     pEc[2][0] = 7780.0;
     pEc[2][1] = 2.00E+11;
     pEc[2][2] = 800.0;
  }
```

A 3 by 3 array of doubles is created and is used to represent the properties of the materials of the problem. The material properties are the density, young's constant and the cost of the material.

2.2.2 Objectives

```
f1 = (Math.PI / ( 2* Math.pow(L,2) )) * (Math.pow((EI / mu),0.5));
     f2 = (2*b*L) * ( (pEc[0][2] * d1) + (pEc[1][2] * (d2 - d1)) +
          (pEc[2][2] * (d3 - d2)) );

     //sets the objective functions that we are trying to mimise
     solution.setObjective(0, f1);
```

The objectives are coded as functions and are set using the solution.setObjective.

2.2.3 Constraints

The constraints were represented using if-then-else statements.

```
   double constraintu = 0.0;
      if( (mu*L) > 2800 ){
         constraintu = (mu*L) - 2800;
      }else if( (mu*L) < 2000){
         constraintu = 2000 - (mu*L);
      }
      solution.setConstraint(0, constraintu);
```

Once the constraint was defined it was added to the solution using solution.setConstraint

2.2.4 New Solution

The values of the variables are set in the 'newSolution()' method

```
Solution solution = new Solution(numberOfVariables,
    numberOfObjectives,numberOfConstraints);


   //d1
      solution.setVariable(0, new RealVariable(0.01, 0.58));
      //d2
      solution.setVariable(1, new RealVariable(0.02, 0.59));
```

New solution created and variables are added using RealVariable which allows the upper and lower bounds to be set.
The upper and lower bounds are defined within the solution. The variable d2 is calculated from what we know about the bounds of the other variables in relation to d2.

2.2.5 Executor

The Executor class is responsible for constructing and executing runs of an algorithm. A single run requires three pieces of information, the problem class, the algorithm used to solve the problem and the number of objective function evaluations allocated to solve the problem.

```
NondominatedPopulation result = new Executor()
.withProblemClass("PPSBProblem")
.withAlgorithm("NSGAII")
.withMaxEvaluations(10000)
.distributeOnAllCores()
.run();
```

3 Algorithm Experiments

The algorithm is run in a 'for loop' for a total of 30 runs. At each run in the for loop the best hyper volume is kept by using an if statement to check if the current best can be replaced with the new run. The best non dominated front from the 30 runs is then plotted and the performance indicators are printed.

```
//check if the new hyper volume is better than the one we currently have
    stored
        if(bestHyperVolume < hyperVolume.evaluate(returnedPop)){
           bestHyperVolume = hyperVolume.evaluate(returnedPop);
           bestNonDominatedPopulation = returnedPop;
        }
```

This if statement checks if the new population has a higher hyper volume than the one we have currently stored, if the hyper volume is larger for the new population we replace it.

3.1 Performance Indicators

To compare and evaluate the performance of the algorithms we need to use performance metrics or indicators. These performance indicators fall into three categories; convergence, spread or both.

Good Pareto optimal solutions should have the following characteristics:

- Minimize the distance of the Pareto front outputted by the algorithm with respect to the true Pareto front.
- Maximize the spread of solutions found by the algorithm.
- Maximize the number of elements of the 'Pareto optimal set' found.

Statistical analyses are provided by the Analyzer. The Analyzer can display the min, median, max and aggregate values for multiple performance indicators, including hypervolume and generational distance.

```
Analyzer analyzer = new Analyzer()
.withProblemClass("PPSBProblem")
.includeAllMetrics()
.showStatisticalSignificance();

Executor executor = new Executor()
.withProblemClass("PPSBProblem")
.withMaxEvaluations(10000);
analyzer.addAll("NSGAII",
    executor.withAlgorithm("NSGAII").runSeeds(30));
analyzer.includeHypervolume();
analyzer.printAnalysis();
```

The above code shows the definition of the analyzer used to get performance indicators of the algorithm. The algorithm is given 30 run seeds.

The measures chosen to evaluate performance in the experiments were

- The hyper volume indicator
  The size of multidimensional space that a set of solutions cover, without counting the same area twice (how much of the objective space is dominated by a given non-dominated set).
- The Generational Distance metric
  The proximity of solutions to the true pareto front.

4 Discussion of Findings

5 Results

Since the pinned pinned beam is a bi-objective problem, printing the results will list the two objective function values for each non-dominated (Pareto optimal) solution. The graphs of the pareto fronts when using these algorithms are below. Both algorithms produce results within the specified solutions. Looking at the graphical representation of the pareto fronts we see that the GDE3 gets a pareto front that initially minimises objective 1 more effectively than than NSGAII however both graphs show the same pareto front shape.

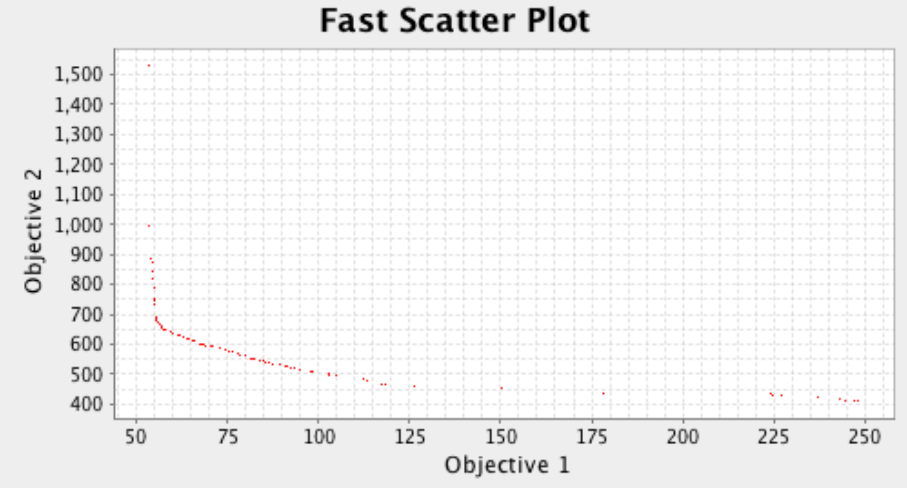

Figure 1: GDE3 Pareto Front Graph

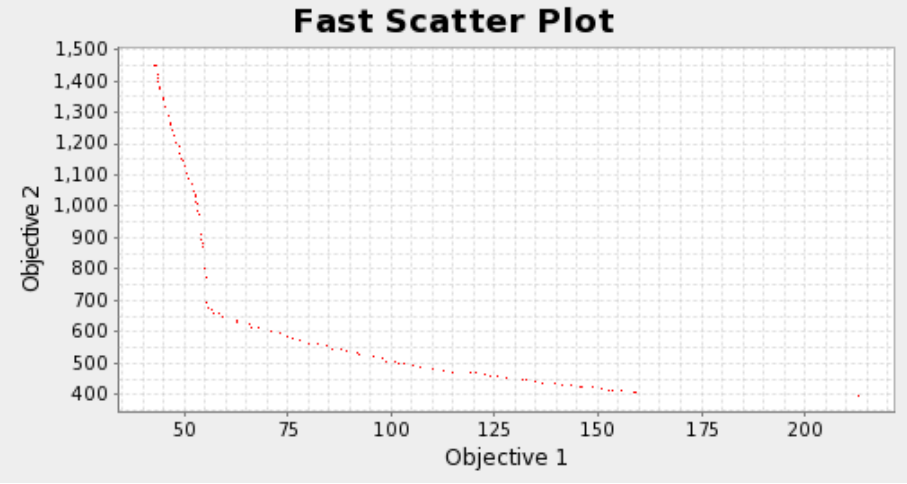

Figure 2: NSGAII Pareto Front Graph

GDE3 and NSGA-II produce similarly large hypervolume values. The values indicate how much of the objective space is dominated by a set of solutions so in this problem a larger hypervolume value indicates a solution closer to the minimum. We can determine that there is no significant difference in performance between GDE3 and NSGA-II for this problem. The Generational Distance is used to measure the proximity of solutions to the true PFs. Therefore we want to minimise this as the closer the results to the true pareto front the better. GDE3 produces a smaller median GD distance of 7.41 compared to 7.99 for NSGAII. However the smallest generational distance was found by NSGAII of 3.02.

Table 1: Table of performance indicators for NSGAII and GDE3

NSGAII	
Hypervolume	
Min	0.5979792476003695
Median	0.8343497204522947
Max	0.8479243002371148
Count	30
Indifferent	[]
GenerationalDistance	
Min	3.0286146382564925E-4
Median	7.990493092788228E-4
Max	0.04202932629391555
Count	30
InvertedGenerationalDistance	
Min	0.007322555675552875
Median	0.017579115461920855
Max	0.18720382686627718
Count	30
GDE3	
Hypervolume	
Min	0.7214580113766463
Median	0.8339981359130433
Max	0.8461024050011707
Count	30
Indifferent	[]
GenerationalDistance	
Min	3.36873603802406E-4
Median	7.415311602000796E-4
Max	0.015424032264177195
Count	30
Indifferent	[]
InvertedGenerationalDistance	
Min	0.010240199774478957
Median	0.024584771930803082
Max	0.20975210303016728
Count	30