Multi-objective optimisation implementation using Java

Shan He

School for Computational Science
University of Birmingham

Module 06-27818 and 27819: Advanced Aspects of Nature-Inspired Search and Optimisation (Ext)

Outline of Topics

1. Existing EA/MOEA libraries and toolboxes
2. Introduction to MOEA Framework
3. How to define a new problem in MOEA
4. How to Evaluate algorithms?

Existing EA/MOEA frameworks and toolboxes

- Matlab
  - Matworks Global Optimisation Toolbox
  - Genetic Algorithm Optimization Toolbox (GAOT)
- Java
  - ECJ: Java Evolutionary Computation Toolkit
  - Watchmaker Framework for Evolutionary Computation
  - JGAP: Java Genetic Algorithm Package
  - MOEA framework ✓
- C/C++
  - ParadisEO
  - Evolving Objects (EO): an Evolutionary Computation Framework
  - Open BEAGLE
- Python
  - PyGMO
  - DEAP
  - inspyred: Bio-inspired Algorithms in Python

What is MOEA framework

- "A free and open source Java library for developing and experimenting with multiobjective evolutionary algorithms (MOEAs)" from MOEA framework web page
- Many state-of-the-art multiobjective evolutionary algorithms are included:
  - Genetic algorithms: NSGA-II and NSGA-III
  - Differential evolution
  - Particle swarm optimization
  - Genetic programming and Grammatical evolution
  - Click here for a list of all algorithms
- Other features:
  - **Easy to use**: minimum effor to define a problem, an algorithm and an experiment
  - **Modular design**: constructing new optimization algorithms from existing components
  - Very good documentation if you pay, but google "MOEA Framework Manual"

Code example 1: Installing MOEA framework (10 mins)

- Easier way:
  - Download the compiled binaries (Version 2.11) from here
  - Unzip and start Eclipse
  - Add all the *jar files into the libraries of your Java Build Path
- More comprehensive way (What we are going to use):
  - Download the source code (Version 2.11) from here
  - Unzip and start Eclipse
  - Select File $\longrightarrow$ Import
  - Select General $\longrightarrow$ Existing Projects into Workspace
  - Select Set Root Directory $\longrightarrow$ Click Browse button
  - Select your MOEAFramework-2.11 folder. Finally, click Finish

How to use MOEA?

General steps:

- **Step 1:** Define a new problem, extend the `AbstractProblem` class
- **Step 2:** Use `Executor`, `Instrumenter` or `Analyzer` classes, depends on your needs:
  - `Executor`: class for constructing and executing runs of an algorithm
  - `Instrumenter`: class for analyzing the performance of algorithms
  - `Analyzer`: class for analysing the resulting Pareto approximation set and how it compares against a known reference set.
- We shall see an example of solving a benchmark test function UF1 using MOEA later

MOEAs Benchamark Test Functions

- ZDT (Zitzler-Deb-Thiele) problems: 6 problems, e.g., ZDT1 - ZDT6
- DTLZ problems: 7 problems, e.g., DTLZ1 - DTLZ7
- WFG problems: 9 problems, e.g., WFG1 - WFG9
- CEC2009 competition problems: 10 problems, e.g., UF1 - UF10
- Other benchmarks problems: LZ, misc, etc.
- All problems are defined in MOEAFramework-2.11 $\longrightarrow$ src $\longrightarrow$ org.moeaframework.problem. $+$ benchmark names
- MOEA Java Doc of the problems

MOEAs Benchamark Test Functions

- Example: CEC2009 competition UF1 :

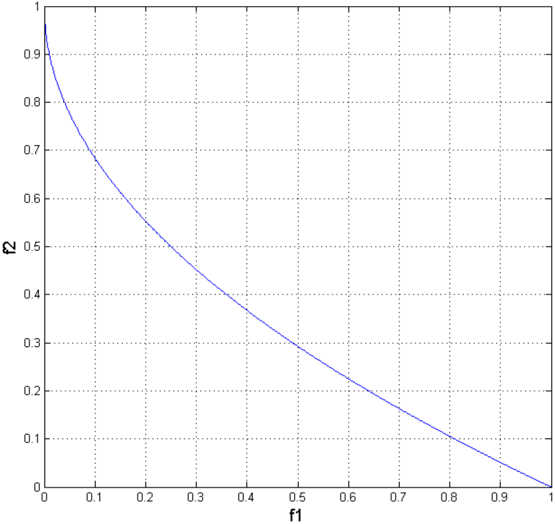
$$\min \begin{cases} f_1(\vec{x}) & = & x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} [x_j - \sin(6\pi x_1 + \frac{j\pi}{n})]^2 \\ f_2(\vec{x}) & = & 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} [x_j - \sin(6\pi x_1 + \frac{j\pi}{n})]^2 \end{cases} \quad (1)$$

where

$$J_1 = \{j | j \text{ is odd and } 2 \leq j \leq n\}$$
$$J_2 = \{j | j \text{ is even and } 2 \leq j \leq n\}$$
$$0 \leq x_1 \leq 1, \ -1 \leq x_i \leq 1, \ i = 2, \cdots, n$$

MOEAs Test Functions: UF1

Pareto Front of UF1: $f_2 = 1 - \sqrt{f_1}$



Code example 2: Solving UF1 problem (5 mins)

We shall execute MOEA to solve UF1:

- Open MOEAFramework-2.11 $\longrightarrow$ examples $\longrightarrow$ (default package)
- Execute `Example1.java`
- Replace line 32 “NSGAII” with “ `Random` ”
- Execute it and compare the results from previous execution
- Explanation of the code

Code Explanation: How to use Executor

- `Executor` class: constructing and executing runs of an algorithm
- Required 3 pieces of information:
  - The problem (line 31)
  - The algorithm used to solve the problem (line 32)
  - the number of objective function evaluations allocated to solve the problem (line 33)
- Use `run` metohd to run this experiment and returns the resulting approximation set (line 35)
- Results, e.g., all Pareto optimal solutions produced by the algorithm during the run are stored in a `NondominatedPopulation` object (line 30)
- In `NondominatedPopulation` , there is an iterable class `Solution` , which stores the decision variables, objectives, constraints and attributes (line 39)

Code Explanation: Customising algorithms

- `Executor` executes algorithms with default parameters
- To set parameters of an algorithm: use `withProperty` (line 34)
- Each algorithm defines its own parameters
- See the API document for parameters of each algorithm

Defining a new problem in MOEA

- All problems in the MOEA Framework is defined by extending the `AbstractProblem` class
- Three things need to be done for defining a problem:
  - Characterizing a problem: number of decision variables and number of objectives
  - Defining the problems solution representation: Are they binary, integer, or real-number? what are their bounds?
  - Evaluation of the solutions: evaluate the solutions using objective functions
- Let's take a look at a new benchmark function DTLZ2

MOEAs Benchamark Test Function DTLZ2

- Defined by Deb ,Thiele, Laumanns and Zitzler (DTLZ) in their paper
- Can have m objectives and n $(n > m)$ decision variables:

$$\min \begin{cases} f_1(\vec{x}) & = & (1 + g(\vec{x}_m)) \prod_{i=1}^{m-1} \cos(x_i \frac{\pi}{2}) \\ f_2(\vec{x}) & = & (1 + g(\vec{x}_m)) \sin(x_{m-1} \frac{\pi}{2}) \prod_{i=1}^{m-2} \cos(x_i \frac{\pi}{2}) \\ \cdots & & \\ f_m(\vec{x}) & = & (1 + g(\vec{x}_m)) \sin(x_1 \frac{\pi}{2}) \end{cases} \tag{2}$$

where $\vec{x}_m$ is a vector of the remaining decision variables x_i, $i = \{m, \cdots, n\}$, and

$$g(\vec{x}_m) = \sum_{x_i \in \vec{x}_m} (x_i - 0.5)^2 \tag{3}$$

and

$$-1 \leq x_i \leq 1, \; i = 1, \cdots, n$$

MOEAs Benchamark Test Function DTLZ2

- For DTLZ2 with $m = 2$ objectives and $n = 3$ $(n > m)$ decision variables:

$$\min \begin{cases} f_1(\vec{x}) & = & (1 + g(\vec{x}_m)) \cos(x_1 \frac{\pi}{2}) \\ f_2(\vec{x}) & = & (1 + g(\vec{x}_m)) \sin(x_1 \frac{\pi}{2}) \end{cases} \tag{4}$$

where $\vec{x}_m$ is a vector of the remaining decision variables x_2 and x_3, and:

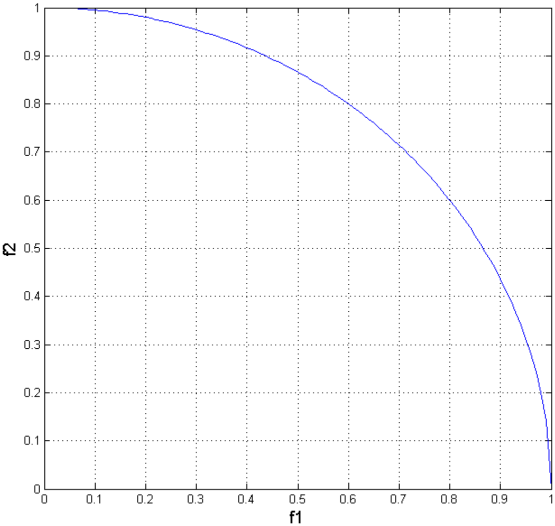
$$g(\vec{x}_m) = (x_2 - 0.5)^2 + (x_3 - 0.5)^2 \tag{5}$$

and

$$-1 \leq x_i \leq 1, \; i = 1, \cdots, n$$

MOEAs Test Functions: DTLZ2

Pareto Front of DTLZ2: $f_2 = \sqrt{1 - f_1^2}$



Code example 3: Defining and solveing DTLZ2 problem (5 mins)

We will execute an example to solve the multi-objective benchmark problem DTLZ2:

- Open MOEAFramework-2.11 $\longrightarrow$ examples $\longrightarrow$ (default package)
- Execute `Example4.java`
- The code lines 70-73 are confusing:
  - Comment out line 70
  - Replace " `numberOfVariables - k` " with " `numberOfObjectives - 1` "
- Explanation of the code

Code Explanation: How to define a new MOO problem

- All problems in the MOEA Framework is defined by extending the `AbstractProblem` class (Line 34)
- Three things need to be done for defining a problem:
  - **Characterizing a problem**: use the constructor function to specify the number of decision variables and number of objectives (Lines 40-42)
  - **Defining the problems solution representation**: override `newSolution` methods in `Soluton` class to define:
    - Use `Solution` constructor to construct new solutions with the number of decision variables and number of objectives (lines 50-51)
    - Use a for-loop to iterate all variables to define variable types (binary, integer, real-number, etc.) and their bounds (lines 53-55)
    - Return the solution (line 57)
  - Evaluation of the solutions

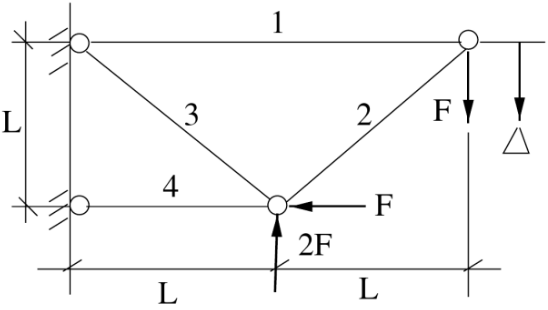
Code Explanation: How to define a new MOO problem

- All problems in the MOEA Framework is defined by extending the `AbstractProblem` class (Line 34)
- Three things need to be done for defining a problem:
  - Characterizing a problem
  - Defining problems solution representation
  - **Evaluation of the solutions:** Implement the `evaluate` method
    - Use `getReal` method in the `EncodingUtils` class to extract the decision variables from a solution (line 67)
    - Instantiate a double array to store the objective function values (line 68)
    - Use those those decision variables to evaluate the DTLZ2 objective functions, e.g., equations (2) - (3) (lines 72-87)
    - Finally we assign the objective function values to the solution (Line 89)

Exercise: multi-objective optimization in structural design

Problem: design of a four bar truss with two objectives:

- Minimise the volume of the truss
- Minimise its joint displacement



Exercise: multi-objective optimization in structural design

Mathematical formulation:

$$\min \begin{cases} f_1(\vec{x}) & = & L(2x_1 + \sqrt{2}x_2 + \sqrt{x_3} + x_4) \\ f_2(\vec{x}) & = & \frac{FL}{E}(\frac{2}{x_1} + \frac{2\sqrt{2}}{x_2} - \frac{2\sqrt{2}}{x_3} + \frac{2}{x_4}) \end{cases}$$

where

$$\begin{array}{l} (F/\sigma) \leq x_1 \leq 3(F/\sigma) \\ \sqrt{2}(F/\sigma) \leq x_2 \leq 3(F/\sigma) \\ \sqrt{2}(F/\sigma) \leq x_3 \leq 3(F/\sigma) \\ (F/\sigma) \leq x_4 \leq 3(F/\sigma) \end{array} \tag{6}$$

where x_i, $i = \{1, 2, 3, 4\}$ are the cross sectional areas of the four bars, the acting force $F = 10$kN, Young's modulus of elasticity $E = 2 \times 10^5$ kN/cm^2, the length $L = 200$cm, the only nonzero stress component $\sigma = 10$kN/cm^2.

Exercise 1: solving the four bar truss design problem (30 mins)

- In your Eclipse MOEAFramework-2.11 project $\longrightarrow$ examples $\longrightarrow$ (default package): right click $\longrightarrow$ New $\longrightarrow$ Class $\longrightarrow$ Type in your class name, e.g., ForBarProblemSolver
- After line 2 `public class ForBarProblemSolver {` , add:

```
public static class FourBar extends AbstractProblem {
}
```

- Move your cursor to those place where Eclipse shows error with read underscore, e.g., `FourBar` , and auto-correct by selecting:
  - Add Constructor ‘FourBar(int, int)’
  - Add Unimplemented Method
- Now you have a template to define the problem. Follow Example 4 to complete the exercise

Exercise 2: visualising Pareto front in Java (10 mins)

- Download my source code and open NSGAIIexample.java
- I will explain how to use `plotParetoFront` class
- Plot the Pareto front of your four bar truss design problem

Multiobjective optimisation performance assessment

- We need to design performance metrics or indicators to evaluate the algorithms' performance.
- What characteristics should good Pareto optimal solutions have:
    - Minimize the distance of the Pareto front produced by the algorithm with respect to the true Pareto front if it is known
    - Maximize the spread of solutions found, so that we can have a as smooth and uniform Pareto front as possible.
    - Maximize the number of elements of the Pareto set (non-dominated solutions) found.
- A myriad of ways of metrics, but can be classified as three kinds for measuring:
    - Convergence: converge to true Pareto front
    - Spread: maintain diversity of solution mainly in objective space
    - Both Convergence and Spread

Code example 4: Performance assessment in MOEA

We shall execute an example to evaluate the performance of your algorithm using `Analyzer` :

- Open MOEAFramework-2.11 $\longrightarrow$ examples $\longrightarrow$ (default package) $\longrightarrow$ `Example2.java`
- Add “, `‘‘NSGAIII"` ” in line 31, just after “ `NSGAII` ”
- Execute `Example2.java`