COMP2002 - Artificial Intelligence

Week 6 - Multi-objective Optimisation

Introduction

The aim of this sheet of exercises is to get you started with implementing machine learning software. You should complete the exercises ahead of the week 7 seminar session. The model answers will be published shortly after, giving you enough time to re-attempt the exercises after the demonstration in the seminar.

Activities

Your task is to go through the following tasks. Please note, you are expected to complete some work on this outside of the timetabled sessions.

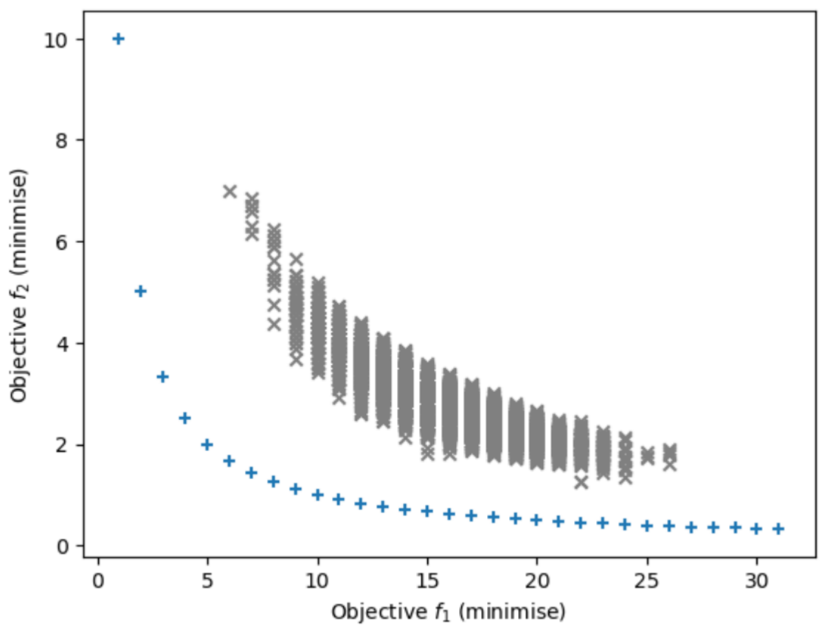
Exercise 1 - Generating random solutions

In last week's coding exercises you worked with binary benchmark problems – **OneMax** and **LeadingOnes**. They were both single-objective, and this week you will extend your code to cope with a multi-objective benchmark problem (**ZDT5**).

I've provided code to run ZDT5, which evaluates given solutions (the **zdt5** function) and also generates the true Pareto front (the **sampleZdt5** function). The first task is to generate the Pareto front, and 10,000 random solutions. Implement the following to do so:

```
import zdt

N = 10000
Y = np.zeros((N, 2))
for i in range(N):
  Y[i] = zdt.zdt5(np.random.randint(0, 2, 80))

plt.scatter(Y[:, 0], Y[:, 1], marker="x", c="gray")

Z = zdt.samplezdt5()
plt.scatter(Z[:, 0], Z[;, 1], marker="+")

plt.show()
```

The result should look something like this – with the Pareto front shown with black + symbols and the random solutions shown by grey circles. The random solutions should be behind (in this case above and to the left) of the Pareto front.

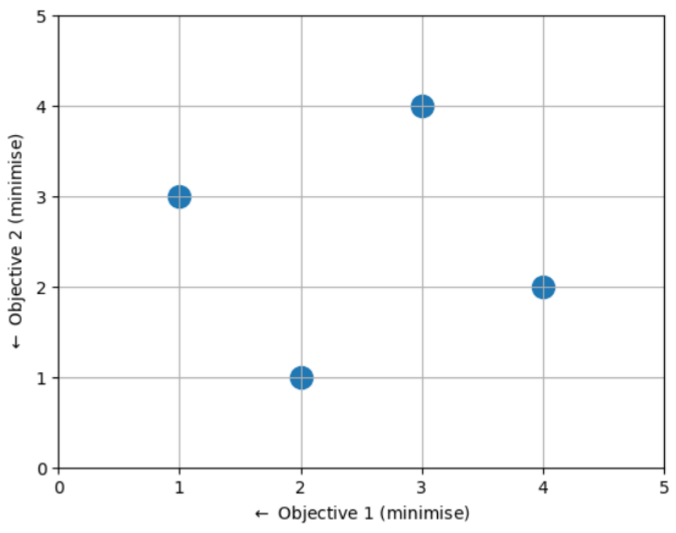


Exercise 2 - Dominance

In order to use your optimiser from last week to optimise multi-objective problems you need to implement the **dominance** relation. Implement a function called **dominates** as follows:

```
def dominates(u, v):
  return (u<=v).all() and (u<v).any()
```

You need to test your function. Use the following multi-objective solutions to test a range of cases – you should check cases where one solution dominates another, and also find a case where two solutions are **mutually non-dominating**.



Once you've implemented **dominates** you can use it to implement a function called **updateArchive**. Your archive should be a Python list, and your **updateArchive** function should take the archive and a new solution's objective vector, and check to see if it should be added to the archive.

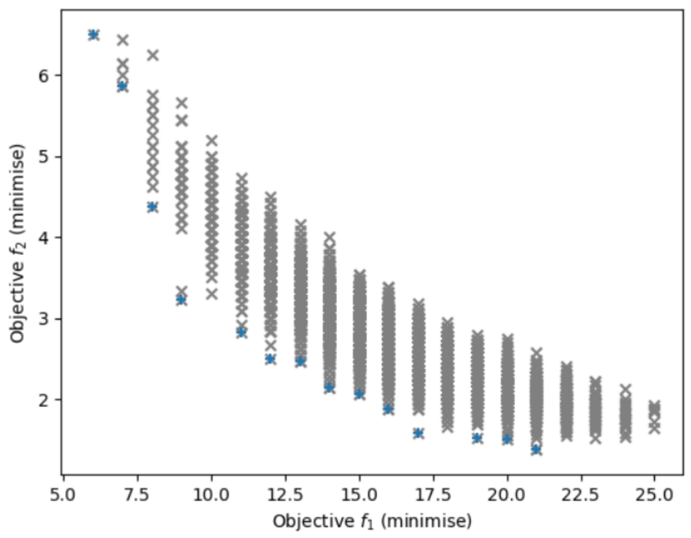
The rules for updating the archive are:

- If the new solution dominates anything in the archive, remove those dominated solutions.
- If the new solution is not dominated by anything in the archive, add the new solution to the archive.

When you've implemented the archive, run the 10,000 random samples through it and plot them against the samples (highlighting the ones in the archive) as follows:

```
# Initialise an empty archive
archive = []

# Add eachof the 10,000 random samples to the archive.
for i in range (N):
  archive = updateArchive(archive, Y[i,:])

# Convert the archive to a Numpy array and plot it.
Z = np.array(archive)
plt.figure()
plt.scatter(Y[:,0], Y[:,1], marker="x", c="gray")
plt.scatter(Z[:,0], Z[:,1], marker="+")
plt.xlabel("Objective $f_1$ (minimise)")
plt.ylabel("Objective $f_2$ (minimise)")
```

You should end up with a plot like this:

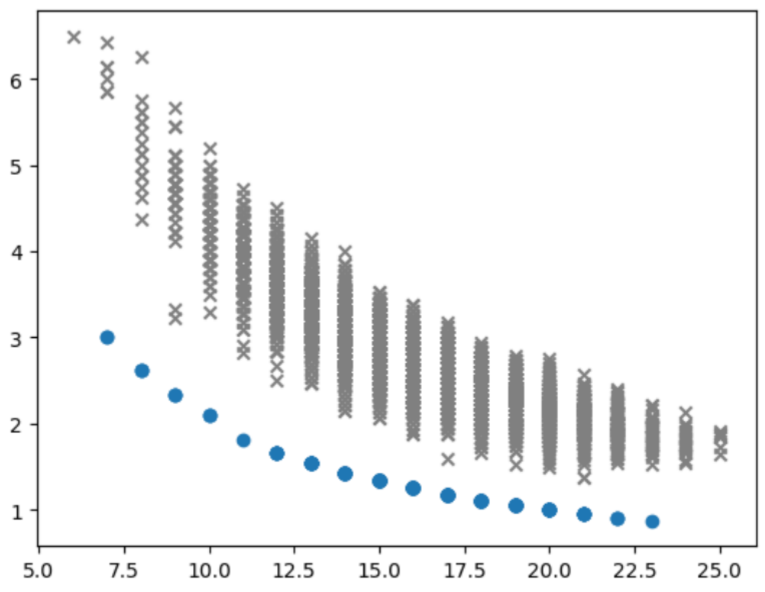


Exercise 3

In order to use last week's optimiser, you need to make a small change to the **evolve** function that will enable you to use the multi-objective archive from Exercise 2. Modify the code so that you call **updateArchive** after you've generated the new child solution, like so:

```
def evolve(x, y, func, mutation, compare, A):
  xp = mutation.mutate(x)
  yp = func(xp)
  A = updateArchive(A, yp)

  if not compre (y, yp):
    return xp, yp, A
  return x, y , A
```

Once you've done that, optimise ZDT5 using the optimiser. Plot the archive against the 10,000 random samples so that you can see how it has improved upon them. Your plot should look like this:



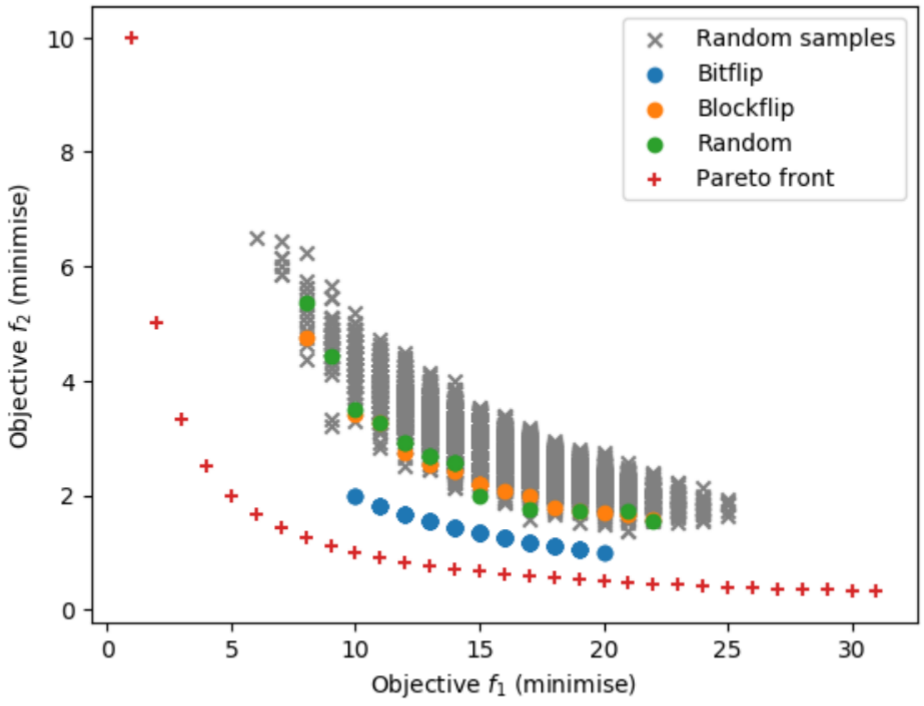
Exercise 4

The final thing to do is to try the different mutation operators from last week on the multi-objective problem. As with last week, include them in a loop so that you produce an archive for each type of mutation.

Produce a plot showing the following:

- The true Pareto front.
- The 10,000 samples.
- The approximated Pareto front for
  - Random mutation;
  - Bitflip mutation;
  - Blockflip mutation.

Your plot should look something like this:



Random samples
Bitflip
Blockflip
Random
Pareto front
Objective f_2 (minimise)
Objective f_1 (minimise)
0
2
4
6
8
10
0
5
10
15
20
25
30