ECTA Homework 4
Multiobjective Optimization with the Non-dominated Sorting Genetic Algorithm II

Arun Prabhu, arun.prabhu@smail.inf.h-brs.de
Dharmin Bakaraniya, dharmin.bakaraniya@smail.inf.h-brs.de

June 28, 2018

1 Assignment Description

1. Implement the NSGA-II algorithm and apply it to a toy problem
   - Bit string with length 20
   - Maximize the number of leading zeros (zeros in a row at the front)
   - Maximize the number of trailing ones (ones in a row at the back)
2. Show that your algorithm works by plotting the population at various stage of the algorithm

2 Submission Instructions

Follow along with the instructions in this PDF, filling in your own code, data, and observations as noted. Your own data should be inserted into the latex code of the PDF and recompiled. All code must be done in MATLAB.

To be perfectly clear we expect two submissions to LEA:

1. 1 PDF (report) a modified version of your submission PDF, with your own code snippets, figures, and responses inserted
2. 1 ZIP (code and data) a `.zip` file containing all code use to run experiments (.m files) *and* resulting data as a `.mat` file
3. 1 GIF (algorithm progress) use the file on the MATLAB file exchange: `https://www.mathworks.com/matlabcentral/fileexchange/63239-gif`

3 The Assignment

3.1 NSGA-II (75pts)

- (50pts) Implement NSGA-II to find all non-dominated solutions to the trailing ones, leading zeros problem.
  - Bitstring with length 20
  - Population size of 100
  - Generations 100
  - Hints:
    - Crossover and mutation can be performed just as in other bit string problems, e.g. one-max
    - The `sortrows` function can be used to sort matrices, you can use this first before implementing NSGAs sorting

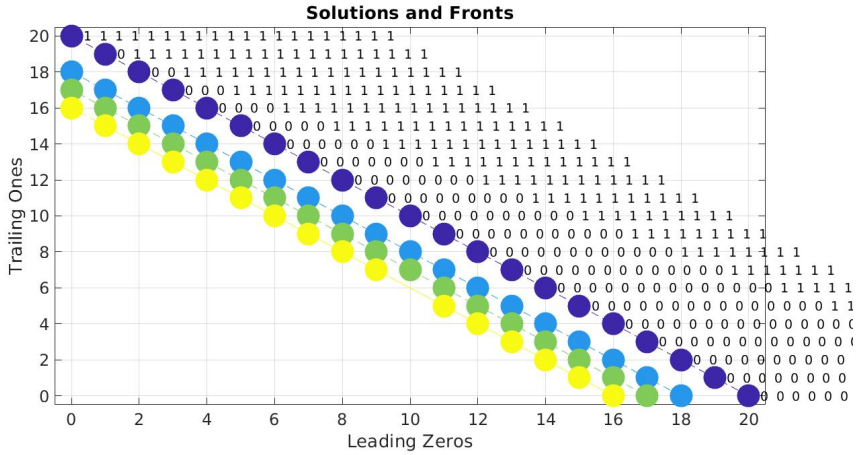

Figure 1: Fronts of NSGA2 algorithm with 2 fitnesses (trailing ones, leading zeros) ran for 100 gen with 100 pop for bitstring of size 20.

- (20pts) Visualize the progress of your algorithm over a single 100 generation run with an animated gif (1 frame every generation).
  - Use the code here: `https://www.mathworks.com/matlabcentral/fileexchange/63239-gif` to create gif
    - Set the timing so that the gif completes in a reasonable amount of time (between 10 and 20 seconds)
  - Fronts can be visualized with the code snippets attached (`displayFronts.m`)
  - Please refer `gif_without_markers.gif`
- (5pts) At each iteration mark the individuals which carry on to the next population, and which do not (you will have to code this yourself).
  - Please refer `gif_with_markers.gif`

3.2 Short Answer (25pts)

- (10pts) Compare the sort used by NSGA-II and MATLAB's quicksort built in `sortrows` function. How long does 100 generations take with each approach when using a population size of:
  - We take average of time taken over 10 experiments.

  1. 10:
     - Quicksort: 0.056817 seconds
     - Fast non-dominated sort: 0.082453 seconds
  2. 100:
     - Quicksort: 0.43277 seconds
     - Fast non-dominated sort: 1.6236 seconds
  3. 1000:
     - Quicksort: 4.9949 seconds
     - Fast non-dominated sort: 111.2765 seconds

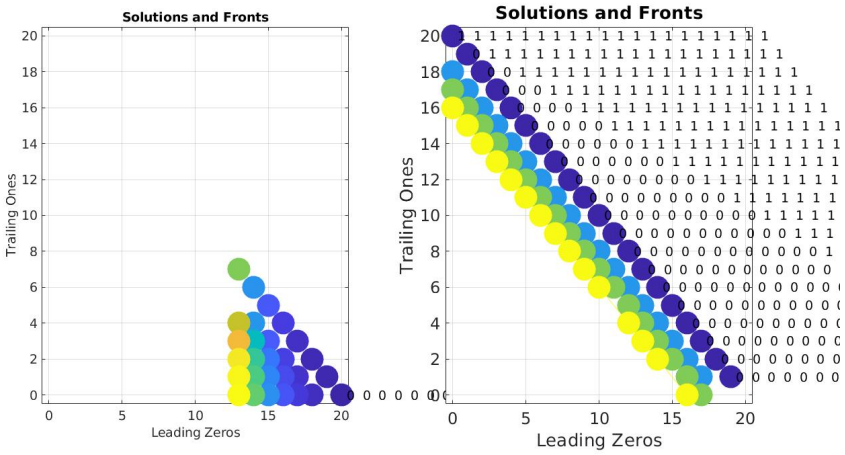

Figure 2: Comparison of quicksort (left) with NSGA2 (right) on 2 functions (leading zeros and trailing ones) for 100 pop and 100 gen on bitstring with size 20 on a single experiment.

  - As it can be seen from Figure 2, quicksort can't even find the front because the sorting is done only with one column. Second column will only be used if first column have a tie. This implicitly gives first function (leading zeros) more importance than second function (trailing ones). This is the reason why we see that most of the population at the end is gathered at the right bottom corner.

- One more thing to note from Figure 2 (left) is that only one individual is in first front.
- Because of the above reasons, we can say that eventhough quicksort takes less time compared to NSGA2, it does not find the solution and thus time should not be the only criteria of performance measurement for these algorithms.

- (5pts) Plot the end result of a single run with [100 pop and 100 gen] and [10 pop and 1000 gen]. Describe the difference between the end results. Which is preferable?

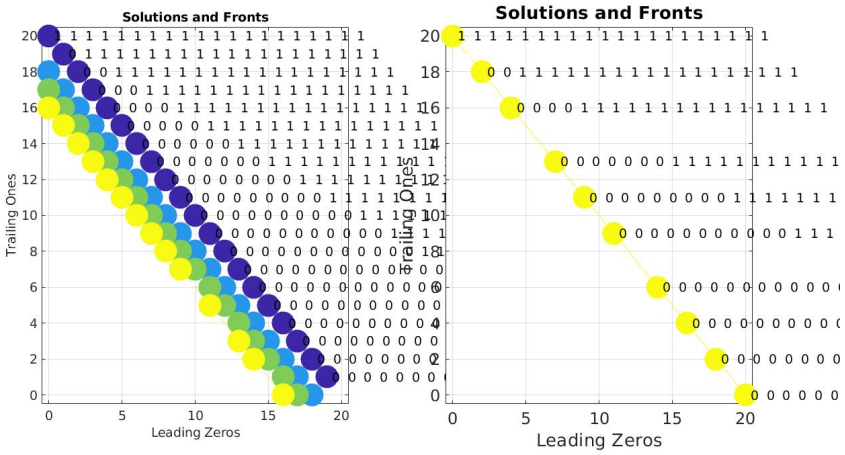

Figure 3: NSGA2 with 100 pop and 100 gen (left) VS. NSGA2 with 10 pop and 1000 gen (right) for solving leading zeros trailing ones problem with bitstring of length 20.

- The `popSize` defines how fine grained solution is.
- As seen from Figure 3, the execution with `popSize=100` (left) produces the complete fine grained pareto front whereas the execution with `popSize=10` (right) reaches the correct pareto front but it is not able to provide the complete front because of lack of individuals.

- (5pts) Imagine you were to replace the objective of "leading zeros" with "largest binary number". Predict the result, and give your reasoning.
  - We expect the first front to have only a single individual with genome of all ones. This is because maximum value of fitness function *trailing ones* would be all ones which is also the maximum for function *largest binary number*.
  - Most of the individuals in any front will be odd number. This is because when the last bit is `1`, the number becomes odd.
  - The even number in a front will occur for values for which the objective function *trailing ones* is 0.
  - As seen from Figure 4, we can confirm our predictions mentioned above.

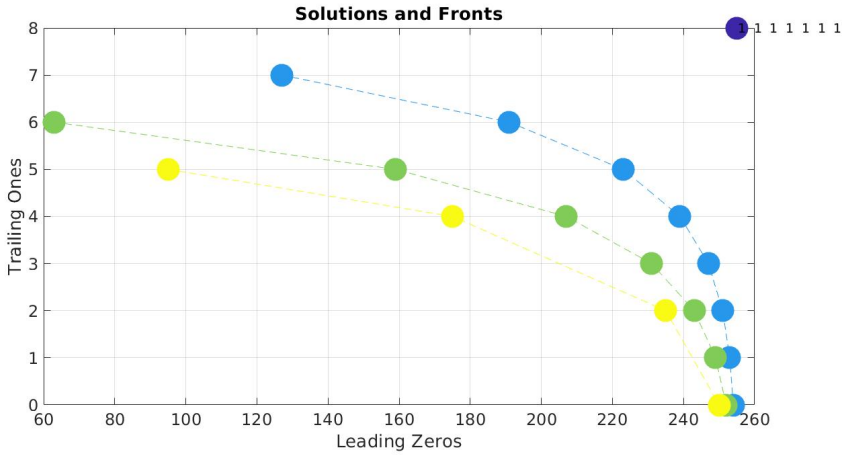

Figure 4: NSGA2 with 20 pop and 20 gen for solving largest binary number and trailing ones problem with bitstring of length 8.

- (5pts) Imagine you were to add a third objective: "non-consecutive ones and zeros" (ones not touching ones and zeros not touching zeros, e.g. 0101 and 1010 are the most optimal 4 bit solutions). How would you adjust the hyperparameters to get a satisfactory result?
  - The `popSize` has to be larger because there is an extra function. This makes the pareto front large, which in turn will need more individual to represent.
  - The `nGeneration` must also be increased because the algorithm will require more time to find and occupy the whole pareto front.

- (5pts) In many GAs and ESs populations must be ranked, but no special methods are used. Why is a faster sort in MOO so important?
  - We have not come across, up till now, any GAs or ESs which required sorting.
  - Because most of the GAs and ESs have single objective function to optimize, when a sorting is required, a conventional sorting is enough, but MOO has multiple objective functions, so when they are sorted, typical sorting algorithms provide biased output (as it is seen from Figure 2)

3.3 ** Extra Credit ** (+ 10pts in examination)

Implement the third objective "non-consecutive ones and zeros"

1. How many non-dominated solutions are possible? (Hint: start with a smaller length and test)

Table 1: Size of non dominated front for different genes

nGenes (n)	size of front 1 (S)	offset (O)
2	3	5
4	8	7
6	15	9
8	24	11
10	35	13

   - $S_{n+2} = S_n + O_n$ and $O_{n+2} = O_n + 2$
   - From the trends seen in Table 1, we predict that the size of the non dominated front solution will be 120 (offset=21) for gene size of 20

2. What changes did you make to the algorithm and hyperparameters to get a good result?

   - We had to change the fitness calculation function and domination calculation functions. Plotting function was also updated to plot 3D plots.
   - We increased the `popSize` from 100 to 150. We also increased the `nGeneration` to 200 from 100.

3. List the solutions in your 1st front. Are they all Pareto optimal? How complete is your front (in percentage, based on #1)

   - Please find the solution in `extraSolution.mat`.
   - We have 120 individuals in first front.
   - From our calculations above, our front is 100% complete.

4. Plot the end result in 3D. (Use `plot3` or `scatter3`)

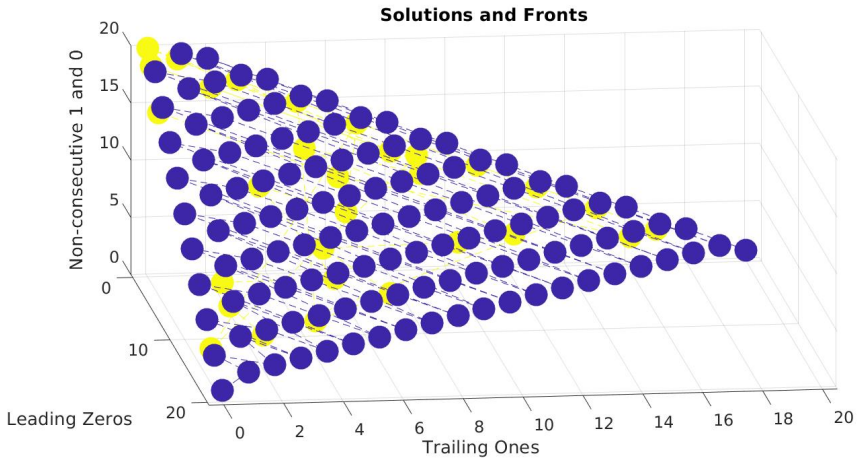

Figure 5: NSGA2 algorithm with 3 fitness (leading zeros, trailing ones and non-consecutive 1 and 0) ran for 200 gen with 150 pop for bitstring of size 20.