Fall 2020 Assignment 1B

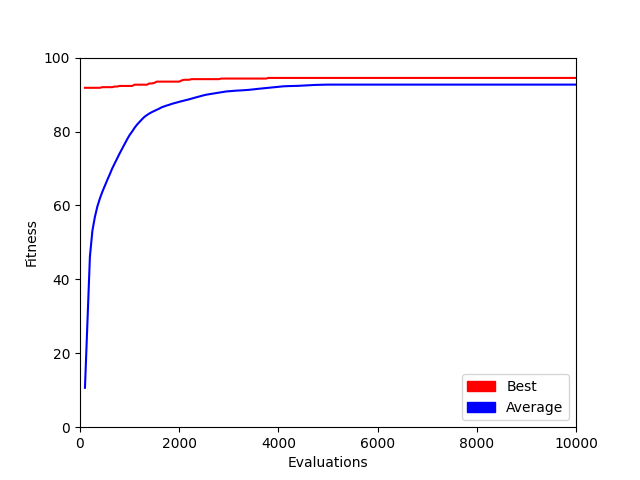
Jordan Sosnowski

[jjs0029@auburn.edu](mailto:jjs0029@auburn.edu)

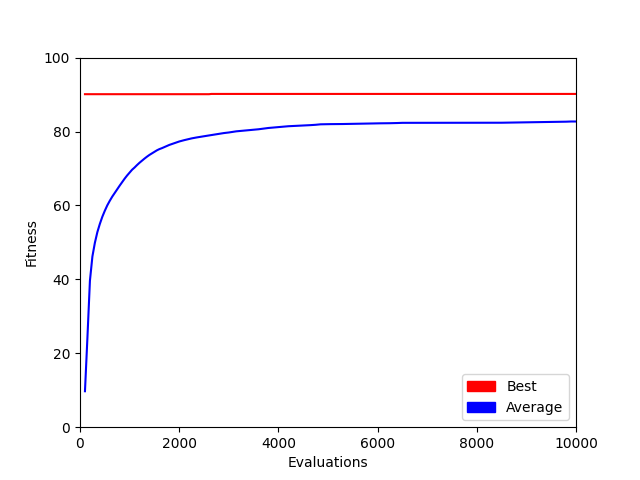
COMP 6660

# Evals vs Average Local Fitness and Evals vs Best Fitness Plots

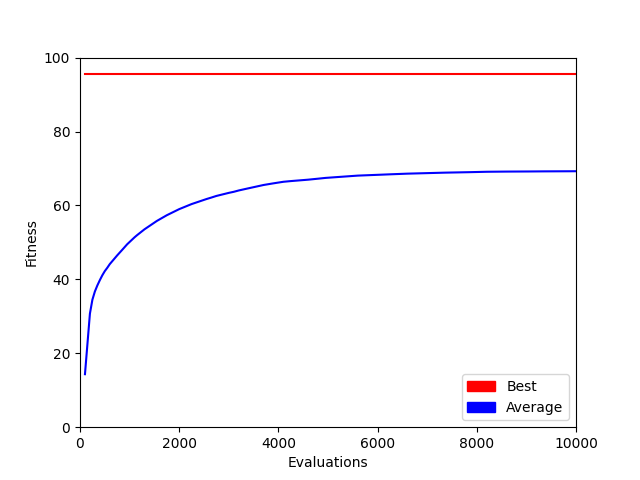
For running our EA against problem A1 we generate the below figure. Since this is an easy problem the graph is what I would expect.



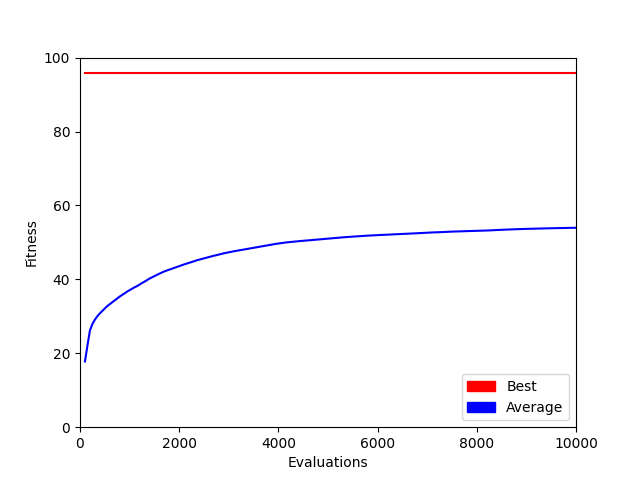
For running our EA against problem A2 we generate the below figure. Similar to A1, A2 is fairly simple albeit a litter harder which explains the lower values and less steep slope.



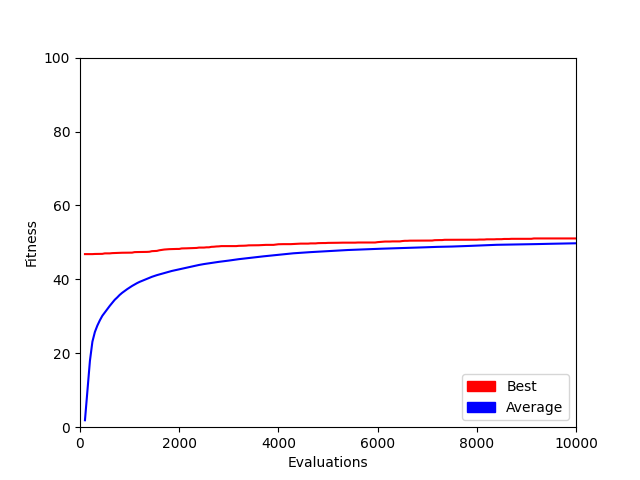
For running our EA against problem A3 we generate the below figure. For the next few graphs, the gap between the best and average fitness is a little startling but I believe it is due to the paring of the truncation survival algorithm and SUS parent algorithm.



For running our EA against problem B1 we generate the below figure.

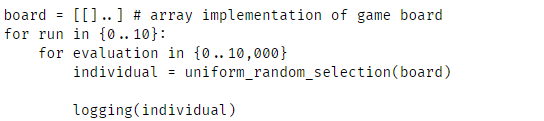


For running our EA against problem B2 we generate the below figure.

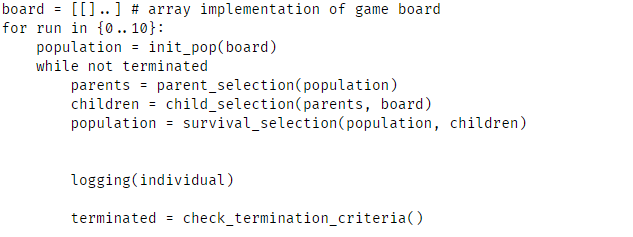


# Statistical Analysis for Random Search and Evolutionary Algorithms for A Problems

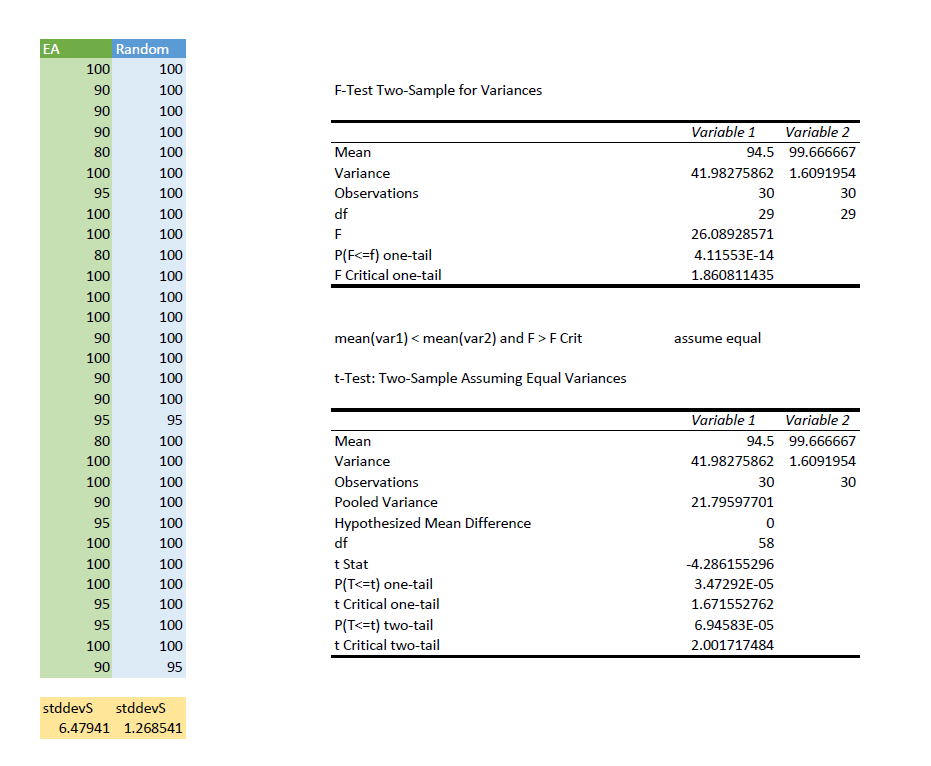
For this section, we were asked to compare the results of the Random Search algorithm against our EA. Below is a pseudo-code implementation of how the random search algorithm would perform. This algorithm simply executes two for loops and for each execution of the inner loops it generates a random possible solution of the board. After the solution is generated it is logged.



Here we have a pseudo-code implementation of how the EA would perform. The main difference between the two is the passing of `genes` between parents and children. Additionally, something I did not realize until later, was that per individual created the evaluation ticker goes up. Therefore, if you are using several evaluations for termination criteria you do not run n generations where n is the max number of evaluations, but n/μ+λ where μ is the number of children produced each run and λ is the number of parents and the number of initial individuals created.

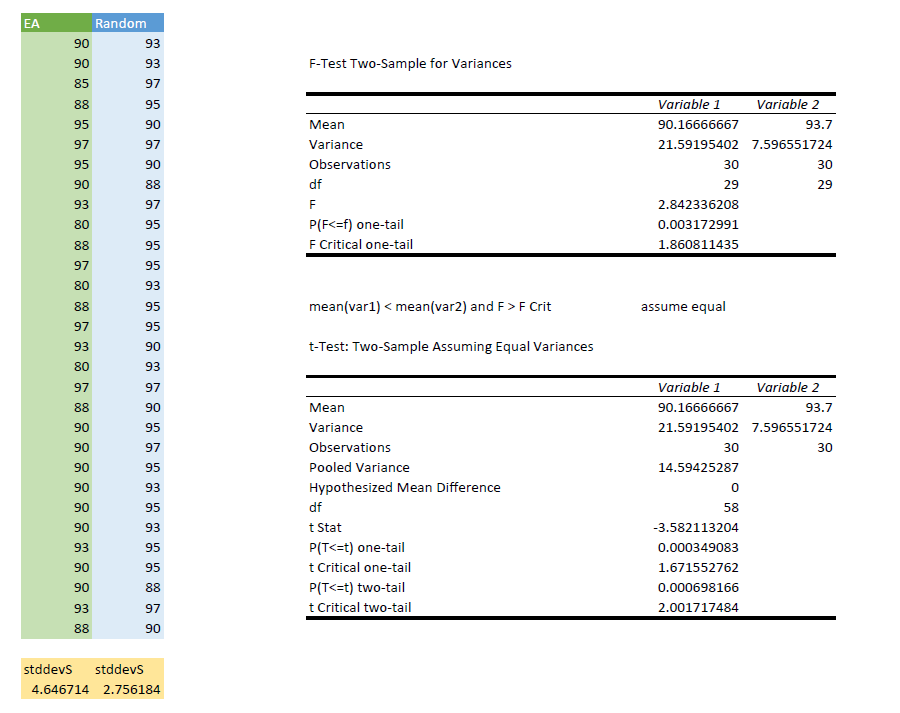


Here we can see the statistical data of running the random algorithm and the EA against problems A1, A2, A3. Both perform quite well on A1 which is to be expected as it is a simple problem.

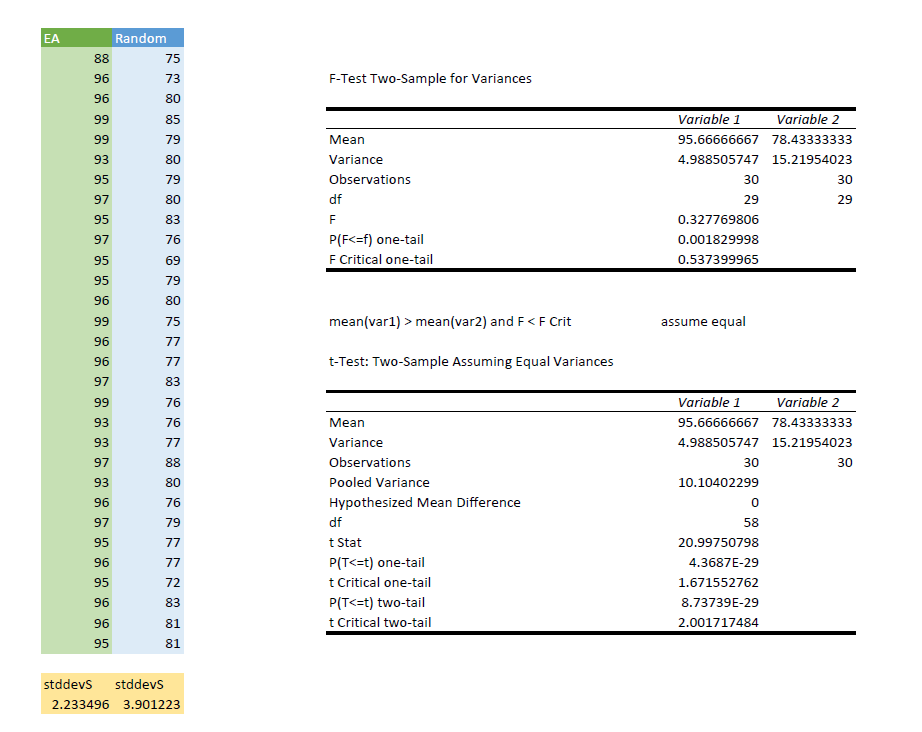


On A2 we see both perform worse than A1, I expected the EA to perform better but due to the random

nature of both algorithms I may have just been a certain experiment that was running.



However, one A3 we see that the EA vastly outperformed the random algorithm which is expected.



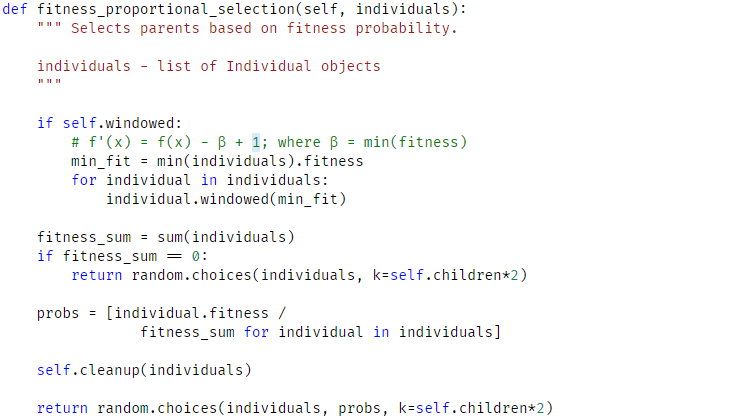
# Stochastic Uniform Sampling vs Fitness Proportional Selection

For the parent selection, we were required to go with three different algorithms: stochastic uniform sampling, fitness proportion selection, and k-tournament selection.

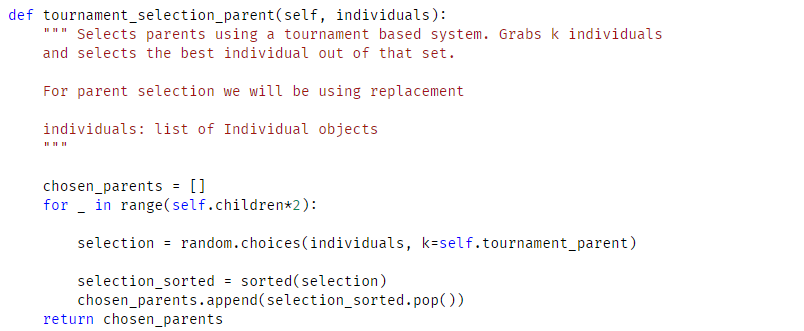
Out of the three SUS was the hardest to wrap my head around. Luckily, the book provided pseudocode for an implementation that I borrowed.



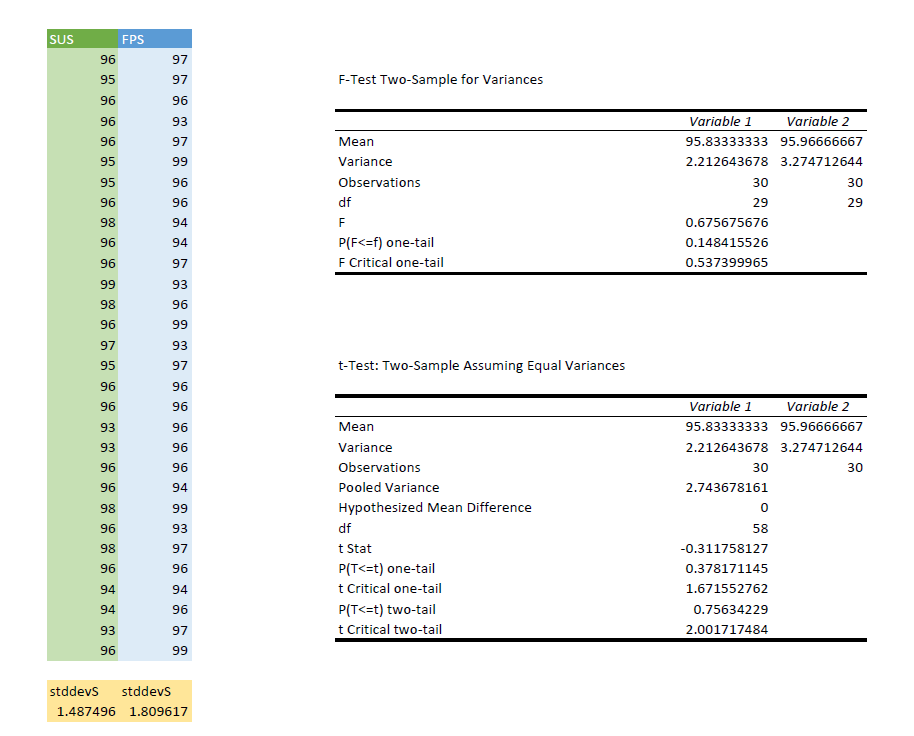
Following this, we have the FPS algorithm which I believe I implemented correctly. The selection for this is based on the absolution fitness of all the individuals. Based on this each individual is selected randomly with replacement.



Lastly, we have tournament selection, due to time constraints I, unfortunately, did not run many experiments with this beyond just initial testing. For this instead of absolute fitness, relative fitness is used.



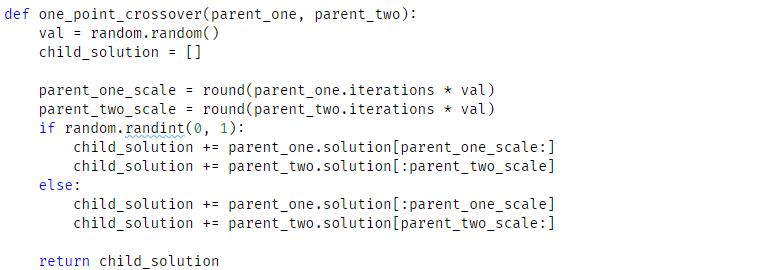
Here we see some statistical analysis for SUS and FPS. Both perform quite well for the problem they ran against which was B1. We can see that FPS has a higher variance than SUS which makes sense based on the biased it can have.



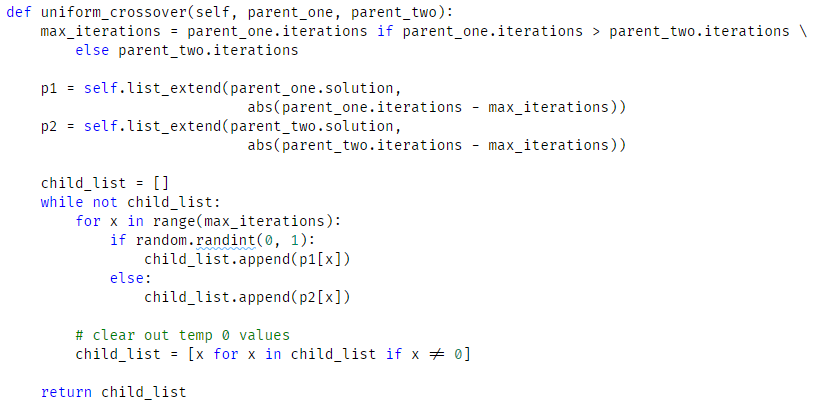
Investigation of Variation Operators

For my framework, I chose initially to go with a one-point crossover as the seemed easiest to implement. I did not realize till later that due to my gene implementation there would not be an equal crossover between parents. In the future, I believe I can solve this in the same way that I used in the uniform crossover function. which will be discussed shortly.

For the one-point crossover implementation, I pick a random value between 0 and 1. This is then multiplied against the length of both parents’ gene pools to scale to each pool. After this, I determine from which parent I should start from and which parent I should end with. Following the merging of the genes, I return the child's solution to be evaluated.



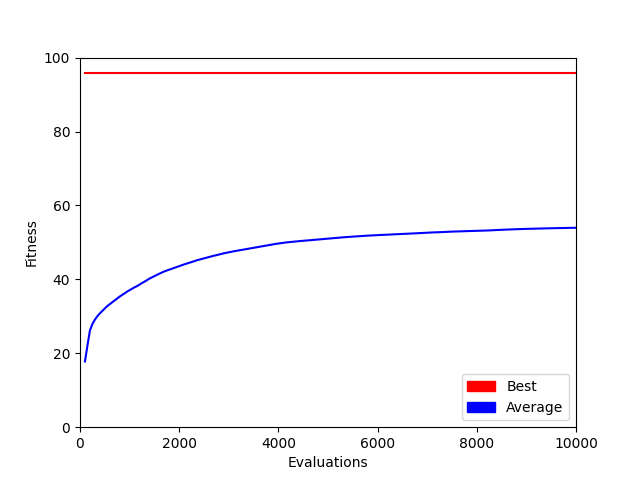
The additional recombination function I created was a uniform crossover implementation. With this instead of splicing the gene pools and selecting a range from each parent, each gene is stochastically determined to pass on to the parent. I first extend the parent's gene pools artificially, so they match in length. Following this, the genes are iterated and with a coin flip, we determine which parent the gene should be inherited from.



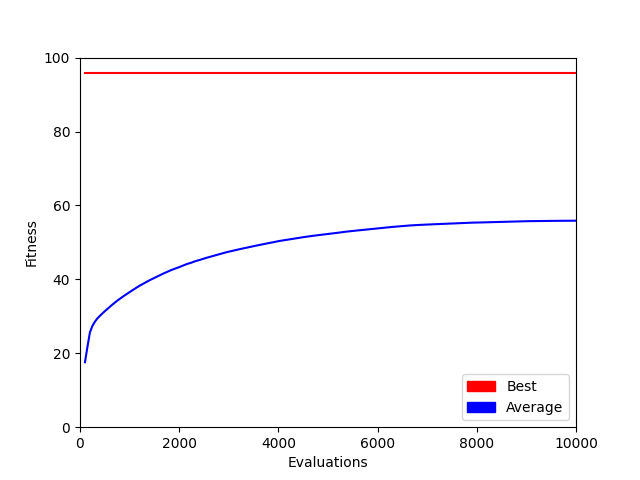
Unfortunately, as seen in the plots below the two implementations do not seem to have a huge effect on overall fitness. I believe this is due mainly to the survivor implementation and not the child selection algorithm. For both runs, I used truncation for the survivor algorithm, and due to running out of time, I was unable to test additional ones with the uniform sampling algorithm.

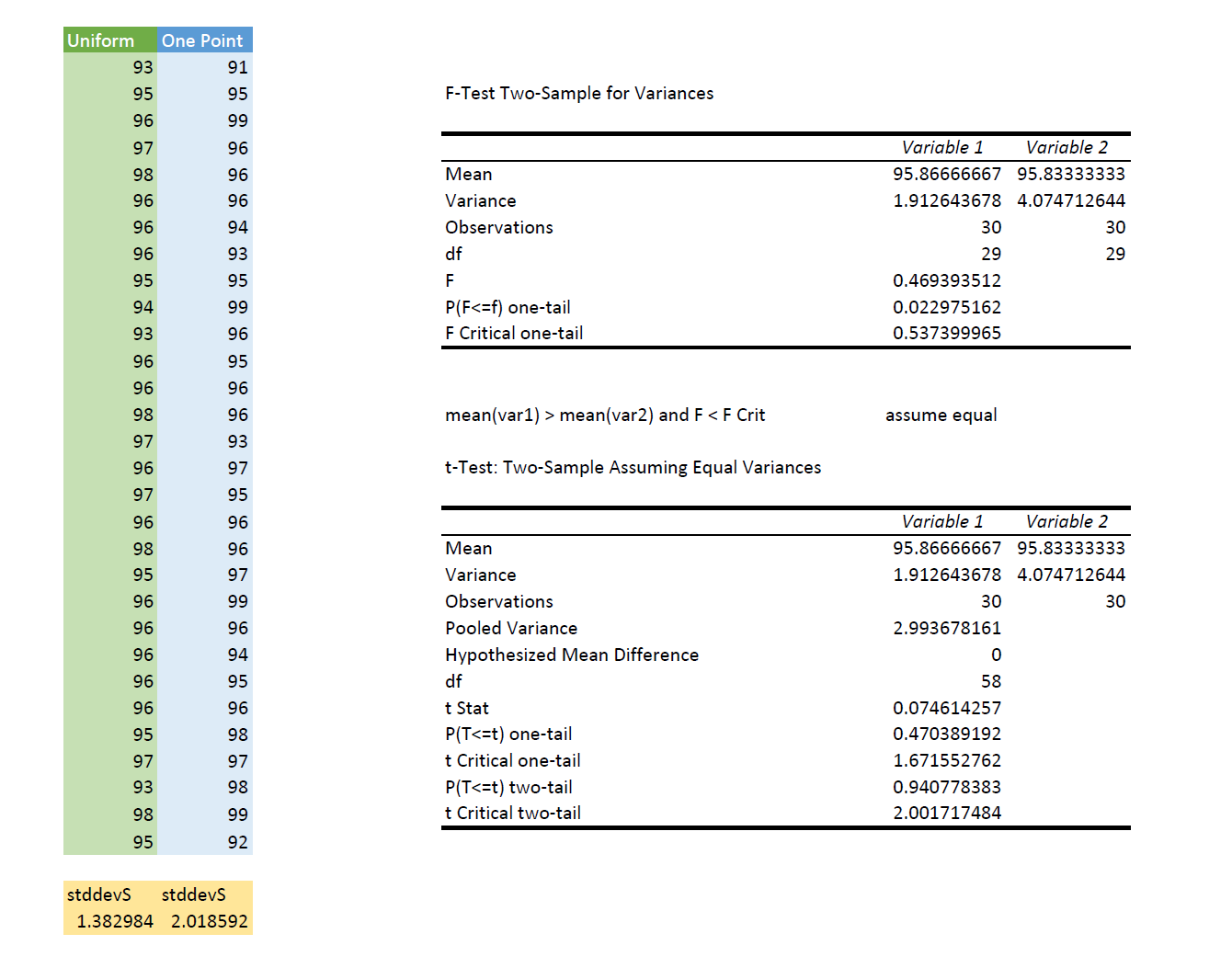
Added with the plots are some statistical analysis. Throughout this project, only problem B2 when comparing uniform against one point produced datasets that were not equal in variance.

B1 One-Point

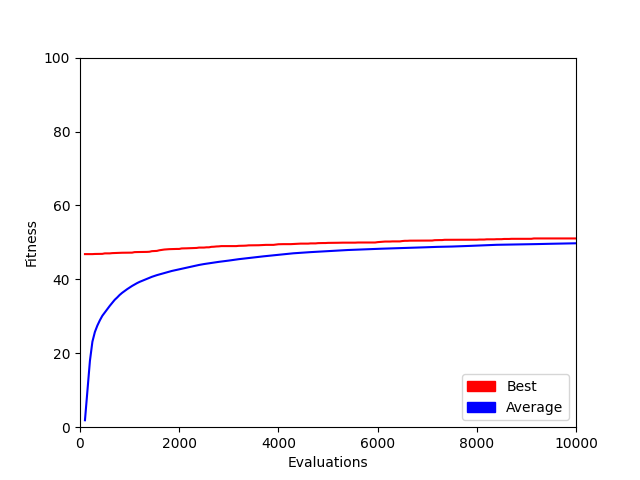


B1 Uniform





B2 One-Point



B2 Uniform

