Garry James McBride

University of Strathclyde

MSc Advanced Computer Science

Multi-Objective Optimisation

Assignment 3: CS547 - Advanced Topics in Software Engineering

Contents

Multi objective optimisation

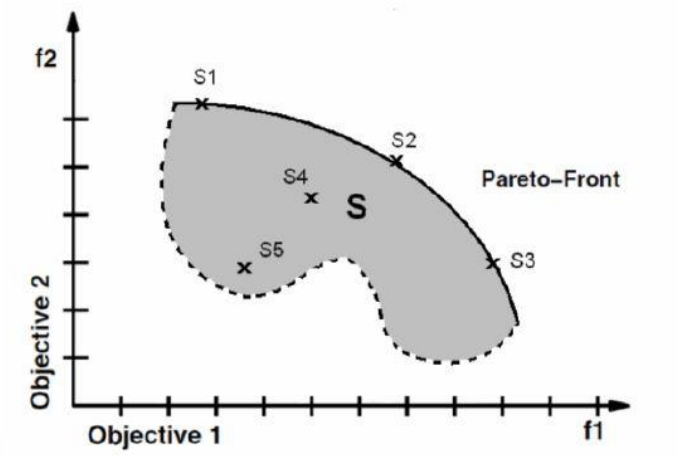
1. Introduction

1.1 Multi-Objective Optimization

Mulit-objective optimization is an area of multiple criteria decision making. Also known as multi-objective programming, vector optimization or Pareto optimization it is concerned with mathematical optimization problems that involve more than one issue/function that must be then optimized simultaneously. Applied to many fields of Science, engineering, economics and logistics, decisions must be made between trade-offs with two or more objectives that are in fact conflicting. The end goal it to minimise the cost whilst maximising the profit for an organization whether it being the purchase of a product, service or other commodity which an organization trades or sells. With regards to practical problems there is most often more then three objectives that must be compared and the best results possible found.

1.2 Pareto Frontier

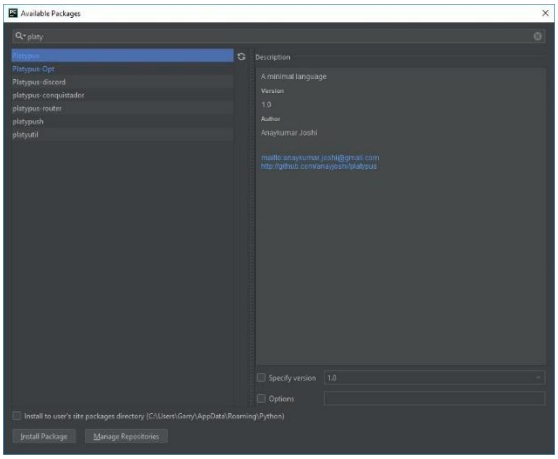
Combined, several optimization objectives are now grouped to achieve the results by software engineers. Real-world software engineering problems can have multiple, competing and contradictory objectives. Maximising and minimising certain aspects to achieve a result or goal, using a pareto optimal (SBSE) which is a state of allocation consisting of resources making it impossible to reallocate without making other individuals or group worse off than they're before state. When presented on a graph, the individuals show a curve, showing their dominance over others below, this is known as the Pareto Frontier or Pareto set.



2. Platypus Package & NSGA-II

2.1 Platypus Installation

The Platypus packages was installed to Pycharm (Jetbrains) via the project interpreter settings.



After installation, the package was imported with the NSGA-II Algorithm and then using the two objective DTLZ2 problem to get used to the package before implementing my own issue/problem for the project. The DTLZ2 is a well-known issue which is included in the package for executing an objective involving two problems as shown in the example below:

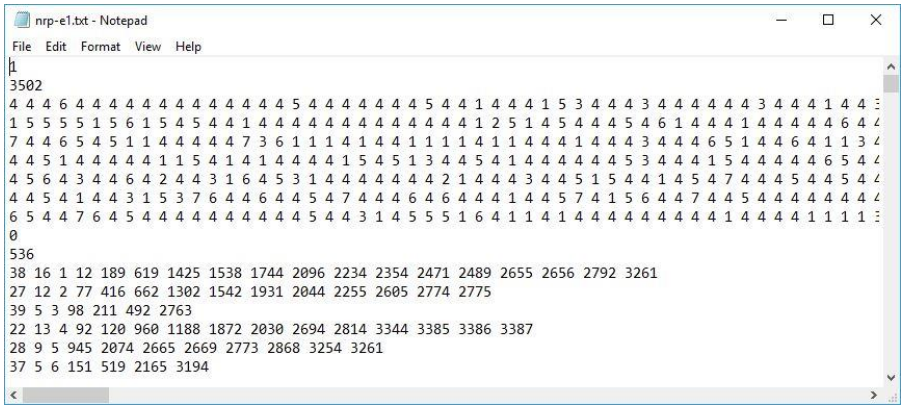
```
from platypus import NSGAII, DTLZ2

# define the problem definition
problem = DTLZ2()

# instantiate the optimization algorithm
algorithm = NSGAII(problem)

# optimize the problem using 10,000 function evaluations
algorithm.run(10000)

# display the results
for solution in algorithm.result:
print(solution.objectives)
```

3. NSGA-II Algorithm

The NSGA Algorithm works by taking all non-dominated (meaning a node is dominated by another i.e.; it is less than and there is no greater node than the better/dominator) solutions from a population after forming a pareto front from a two-objective problem the population becomes less than that of its initial size and uses the nodes to allow the pareto front to then become the remaining number of the population. After this, mutation and cross-bred come into play with the remining population now discarded to form a new population, repeated over *n* of iterations until no solution exists which is dominated by each other in the new population or new generation.

3.1 Understanding the data

The data has been given in notepad documents. Two Zip folders named ‘classic-nrp’ & ‘realistic-nrp’, both included with multiple files to choose from along with example and readme files.



Number	***Number Objective***
1	*Just an exact number, 1*
2246	Number of requirements in Level 1
6 4 4 6 …	*Costs of requirements*
0	*Just an exact* number, 0
294	*Number of customers*

With the irrelevance of those highlighted in Red, they were removed, and the table below was put into a separate folder, with the customer costs being kept in another folder so they may interlink, they are the most important aspects for the issue at hand.

Profit of Customers	***Number of requests by Customer***	***Requirements List***
35	8	1 79 295 513 514 535 971 1736
32	10	2 484 756 994 995 1795 1796 1817 2113 2185
18	5	3 209 561 714 2136
39	18	4 6 32 93 128 264 603 783 786 992 1012 1030 1213 1217 1325 1423 1780 2016
34	7	7 675 790 807 877 2138 2186

4. Implementation

4.1 NSGA-II

The NSGA-II has been designed to never exceed/minimize the cost whilst maximizing the value/profit, which is the goal of an organization producing such an algorithm for this function. The algorithm was run for 10,000 iterations with a budget of 1, and after the results were shown and the time taken to produce said results, the iterations were experimented with by being dropped to 1000 & 750 with a budget of 0.5. Doing this would allow a good comparison and experimentation of results for the best possible solutions. Time is a key factor in any organization, it was interesting to see how negative/positive the results would return when taking less time to produce the algorithm.

4.2 Single-Objective

The single-Objective algorithm was run with the same method as the NSGA-II on both data sets for experimental purposes of changing iteration and budget, continuing with the time factor to see the difference in results with more/less of said time. Both methods subs classed to the "PurchaserCustomer" class, they both use different weightings (0.5, 0.1, 1) to produce a result composed of both value and cost for the algorithm.

4.3 Random Search

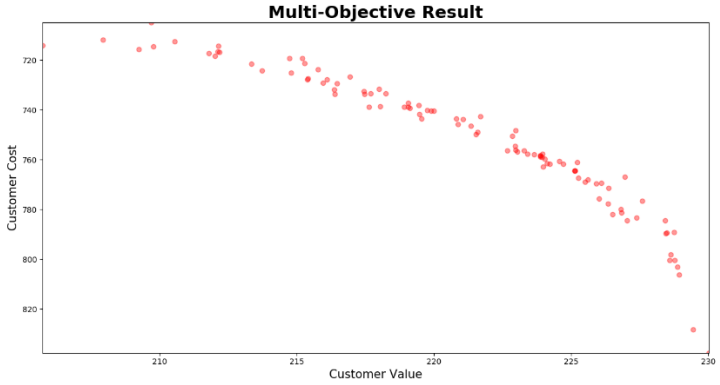
The random search will use the multiple-objective search class (since this is project is based on this method), with a random population generated using different iteration amounts so a good comparison can be evaluated. Not cross-bred or mutated, it has been given the same amount of attention as the other algorithms (which use the functions random search does not) for a very accurate result to compare.

5. Results

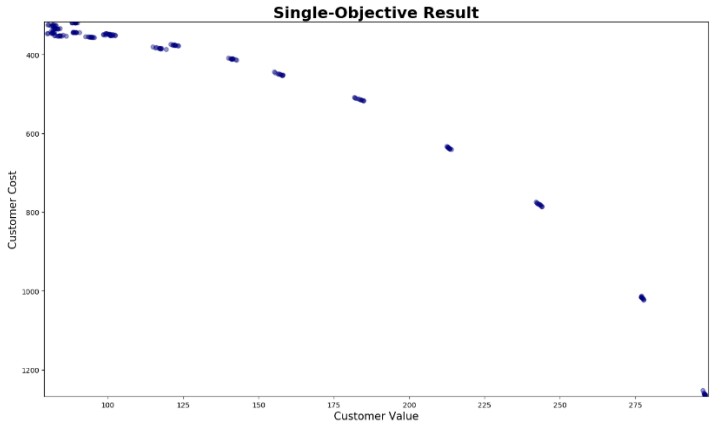
5.1 Realistic Data Set

Iterations: Multi-Objective = 10,000, Single-Objective = 10,000, Budget = 1, Random = 100

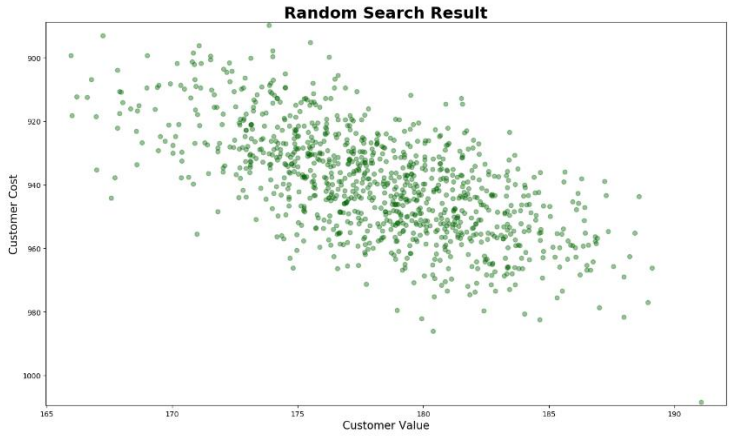
5.1.1 Multi-Objective NSGA-II



5.1.2 Single-Objective Genetic Algorithm

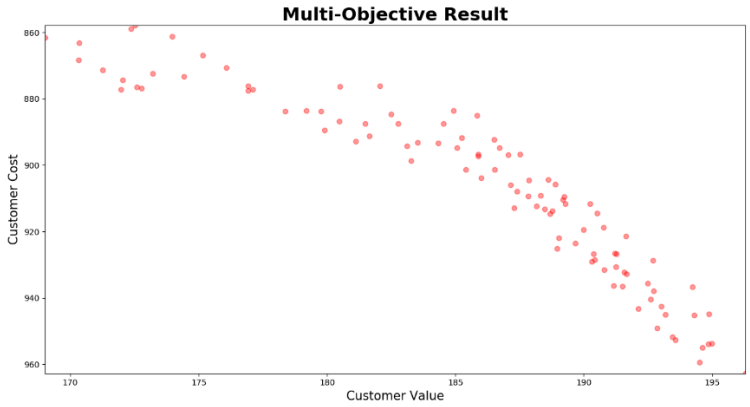


5.1.3 Random Search

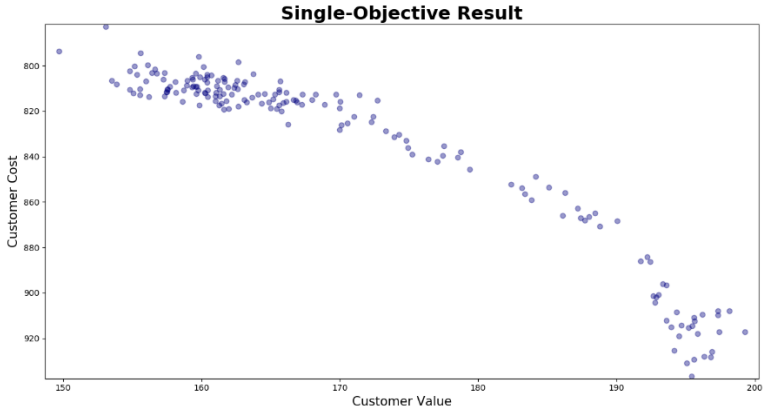


Iterations: Multi-Objective = 1000, Single-Objective = 1000, Budget = 0.5, Random = 50

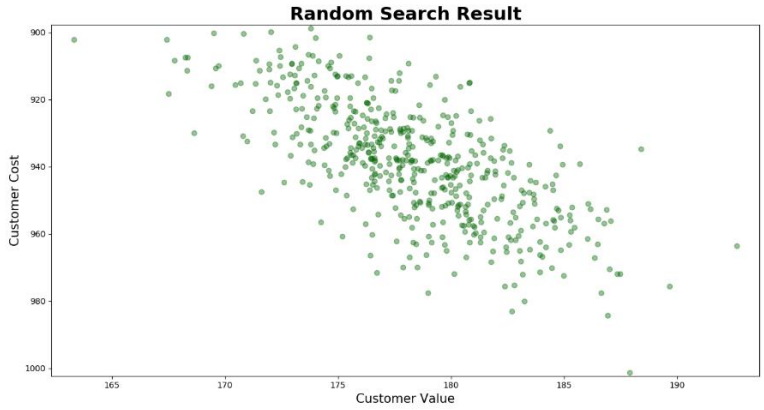
*5..1.4 Multi-Objective **NSGA-II***



5.1.5 Single-Objective Genetic Algorithm

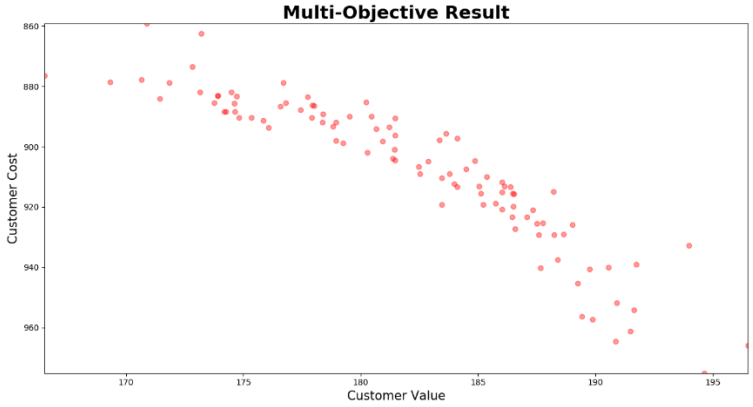


5.1.6 Random Search

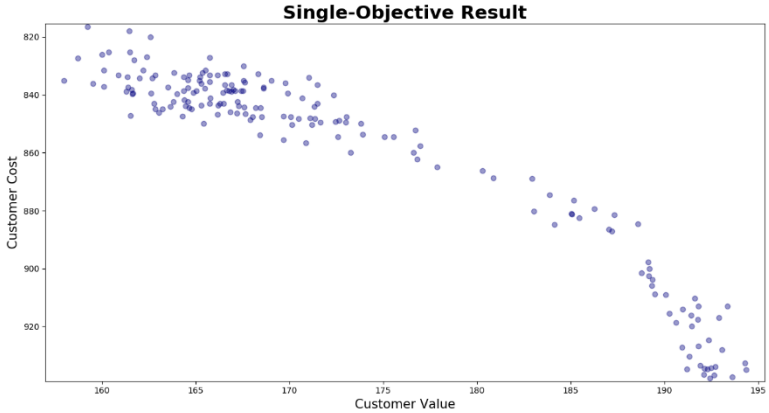


Iterations: Multi-Objective = 750, Single-Objective = 750, Budget = 0.5, Random = 25

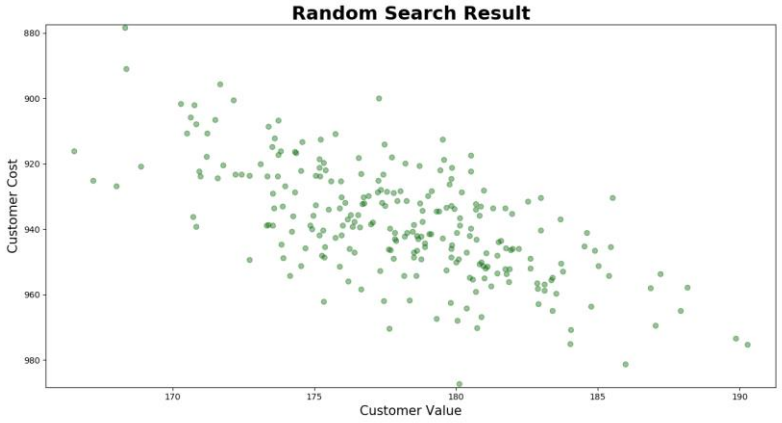
5.1.7 Multi-Objective NSGA-II



5.1. 8 Single-Objective Genetic Algorithm



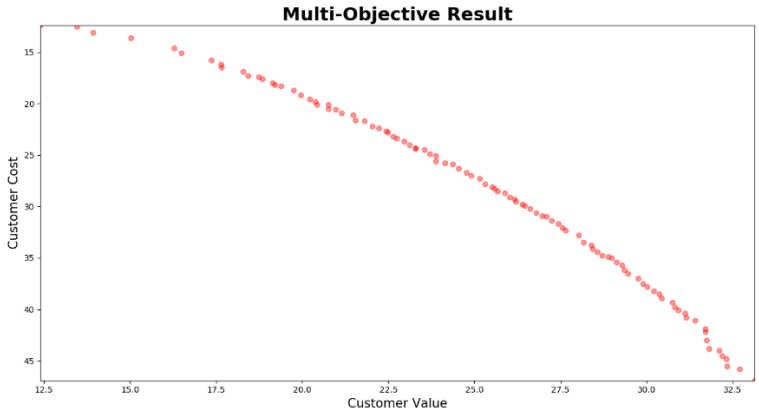
5.1.9 Random Search



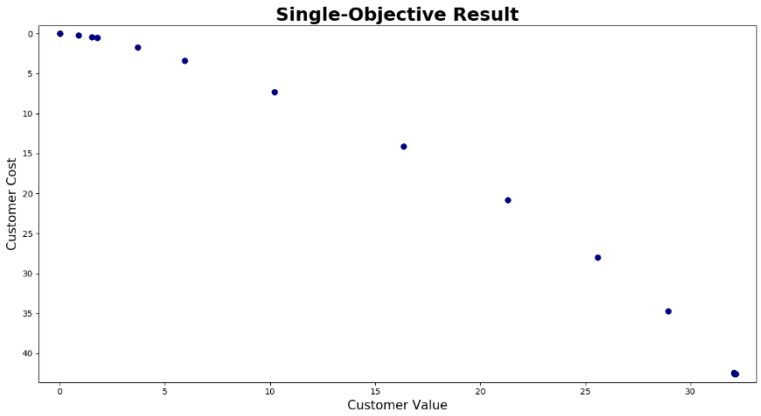
5.2 Classic Data Set

Iterations: Multi-Objective = 10,000, Single-Objective = 10,000, Budget = 1, Random = 100

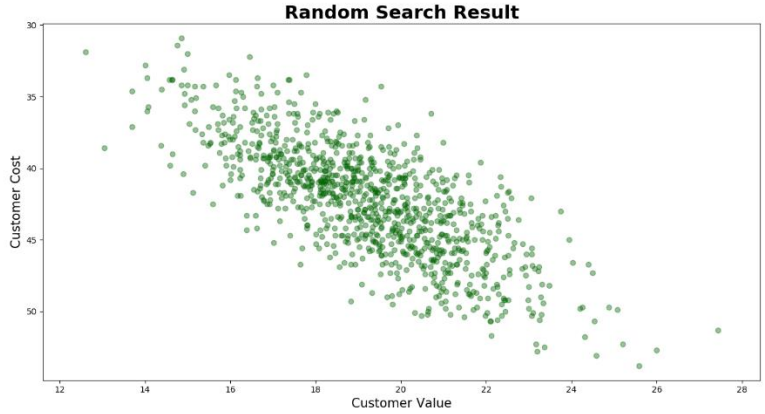
*5.2.1 Multi-Objective **NSGA-II***



5.2.2 Single-Objective Genetic Algorithm



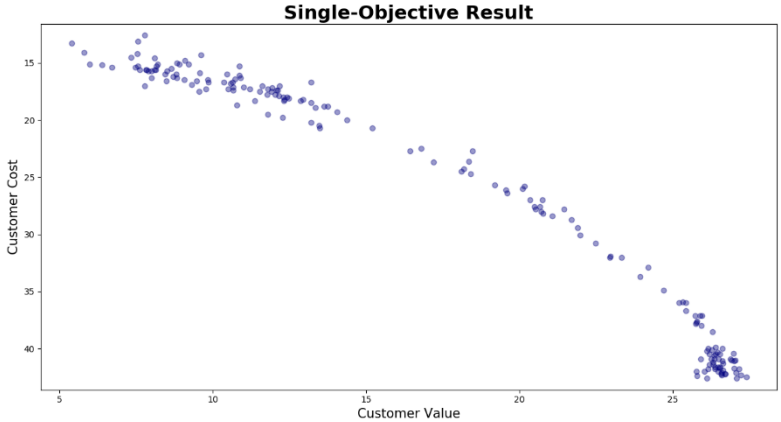
5.2.3 Random Search



Iterations: Multi-Objective = 1000, Single-Objective = 1000, Budget = 0.5, Random = 50

5.2.4 Multi-Objective NSGA-II

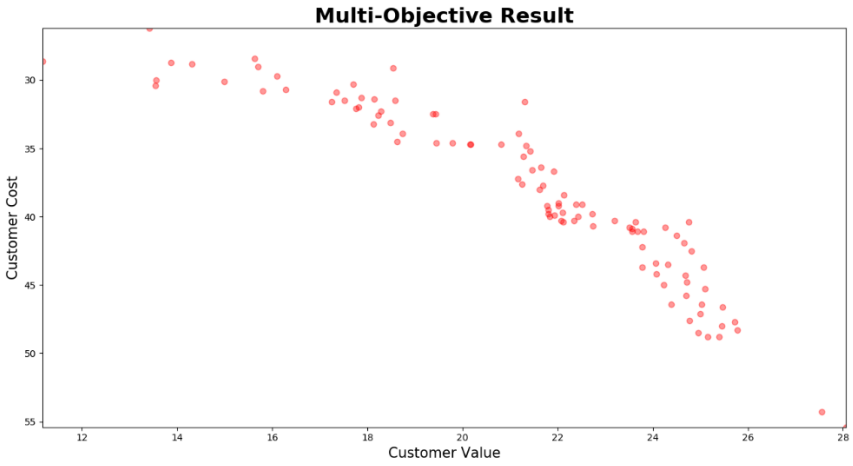
5.2.5 Single-Objective Genetic Algorithm



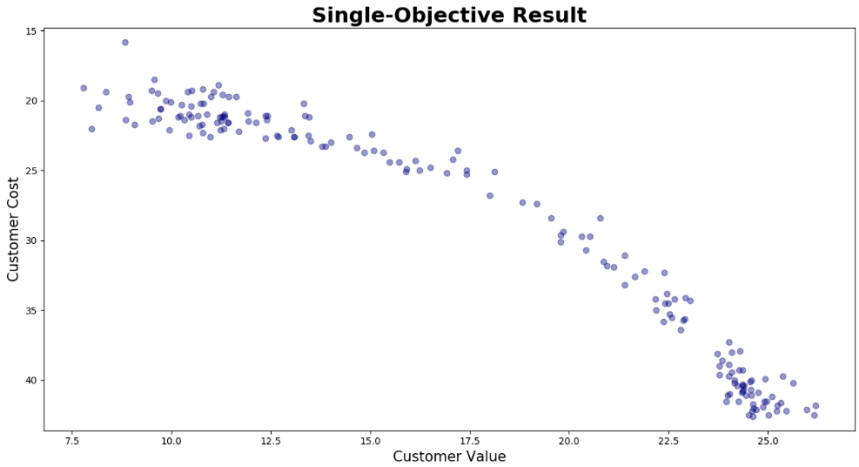
5.2.6 Random Search

Iterations: Multi-Objective = 750, Single-Objective = 750, Budget = 0.5, Random = 25

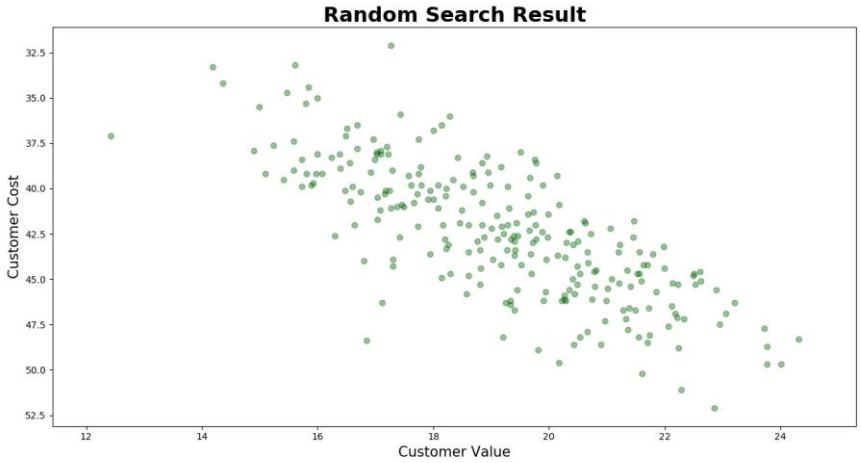
5.2.7 Multi-Objective NSGA-II



5.2.8 Single-Objective Genetic Algorithm



5.2.9 Random Search



6. Analysis

6.1 Realistic Data Set

6.1.1 Multi-Objective NSGA-II

The Results from the NSGA-II Algorithm took 1 hour to run before producing a Pareto Front after 10,000 iterations, not very defined or having a smooth curve, compared to the run of 1000 iterations which took a 1/10 of the time to run. The more iterations the better the pareto front, as also shown and proved in the 750-iteration run where the pareto front appears more inconsistent with the curve.

6.1.2 Single-Objective Genetic Algorithm

The single-objective algorithm took more than 3 hours to run with 10,000 iterations, showing a cluster in solutions spread quite far apart but have shown a very positive curve. Running for a 1/10 of the time of the first result, the pareto front seems less clustered but solutions have dropped into a near

dominated position, the curve has disappeared with fewer iterations, and similar results when the iterations were dropped to 750 iterations, this is could be due to the budget being cut from 1 to 0.5.

6.1.3 Random Search

The random search at 10,000 iterations was not a effective as single or the NSGA-II (Multi-objective) as the iterations were dropped to 1000 and then 750, the random search appears to have stayed in the same position but with less solutions found, making it clear the longer it is run, the results do not improve.

6.2 Classic Data Set

6.2.1 Multi-Objective NSGA-II

The classic data set (much smaller than realistic) appears to have formed a very smooth curve with a cluster of nodes together at 10,000 iterations. When dropped to 1/10 of of initial iteration size, the results show a more spread out amount of solutions at a much lower number than the realistic data set and the curve has become less obvious although still present. The same can be said for the iteration result using 750, the curve is still present but not as smooth as 10,000 iterations showing the longer run time the better result.

6.2.2 Single-Objective Genetic Algorithm

The single-objective algorithm shows a spread out but very dense selection of solutions which has a very smooth curve, clustering is highlighted using transparency and the darkness of solutions beside or near each other. A much faster data set to run because of the size difference, when dropped to 1000/750 iterations, and the drop-in half of the budget, the pareto curve is still present and is not as curved as the other result.

6.2.3 Random Search

Finally, the random search on the classic data set is similar to the realistic data set, where the drop-in iterations (10,000, 1000, 750) show no pareto front, only a more dense set of solutions, clustered and a large amount of domination is present. Random search has proven to produce the same results despite the drop-in iterations and the budget cut.