Next Release Problem

Mihail Mihaylov

Reg. No: 201888925

Introduction

This report includes a description of the next release problem (NRP) as well as implementation details using a multi-function and a single-function objective genetic algorithm. In more details this report would focus on data analysation and representation of the solutions, how the fitness functions were calculated, the algorithm used for crossover, the OPT4J Framework as well as the results yielded from the genetic algorithm and comparison with random search. The next release problem solution is used to determine from a list of requirements what the next update of the software will contain depending on the cost of the implementation of these requirements and the value that customers of the company hold over them. The solution is obtained using the OPT4J Framework and a genetic algorithm with the NSGA-II algorithm to determine the crossover. More details of the implementation are described below.

Key Aspects

Data Analysation

The data supplied was in the form of a text file (.txt), which contained a set of requirements and associated costs on different levels as well as dependencies which were ignored for this project and a list of customers with their values to the company and a list of requirements that they wanted implemented in the next release. The data was analysed using a buffered reader and it was stored in 2 separate lists. The first list was a HashMap with an integer and double variables in it. This map stored the list of requirements and the associated costs. The second list was a list of customers holding Customer objects which contained the value of the customer and the list of their requirements. Furthermore the total cost of all the requirements was summed and each requirement was divided by that sum to provide a percentage of the total cost that each requirement holds. The same thing was done to the value of the customers. They were all summed and each value was divided by the total and stored in place of the original value they held. Once those two lists were obtained there was a for loop implement to go through each of the customers and check the requirements that they have requested. To store this data another HashMap was used. The key of the map was the number of the requirement and the value was the value this requirement held to the customers of the company. When the for loop goes through all the customers it checks the requirements that they have requested and adds the value of the customer divided by the total number of requirements from that customer to the value of each requirements. Thus after analysing the data the results returned are 2 HashMap that held each requirement and their cost and value.

Representation

The representation of the solutions was done using a Boolean genotype in the GACreator class with the total number of requirements. It marked the requirements which would appear in the solution with a “1” and the ones that don’t with a “0”. The selection of 1s and 0s for the genotype is chosen randomly. This was then converted to a phenotype which was a list of integers. A for loop

implemented in the GADecoder class goes through the genotype and if there is a “1” it adds the number of the requirement to the list of Integers.

Fitness Function

Multi-function Objective

The multi-function objective in this case contains two functions. One is the total cost of the requirements in a solution and the other is the total value of the requirements in that solution. Using a genetic algorithm yields an optimized list of solutions (a Pareto front) which are solutions who have the least cost for the maximum value. The total cost and value of the solutions is obtained using the 2 HashMap from the data analysation and the phenotype from the representation. A for loop goes through the phenotype and checks the requirement numbers in a solution and then the requirement’s cost and value were obtained from the 2 HashMaps and added to a total value for each solution. Those were then added to the Objectives used to determine the fitness of the solutions. This part is implemented in the GAEvaluator class.

Single-function Objective

The single-function objective is obtained using a formula which uses both the cost and the value from the multi-function objective to calculate a single score for each solution. The formula is:

$$single = w * value + (1-w) * cost$$

The variable w in this formula is a number between 0.1 and 0.9 which is used to show whether the value or the cost should be more important to the single function score. There is also a variable which is called maxCost which is the maximum cost a solution can reach to be a viable solution. This maxCost is divided by the total cost of all the requirements in order to change it into the same format as the rest of the costs of the solutions. In the case that a solutions total cost is more than the maximum cost declared the total cost of the solution is set to -1 which puts this solution at the end of the population and gets it removed from the next generation. This is also implemented in the GAEvaluator class.

OPT4J Framework

The OPT4J Framework was used for the implementation of this project. More specifically, the creator methods were used to create a Boolean genotype which is also part of that framework for the representation of the solution. Then the decoder class from the framework was used to create a phenotype of the genotype of the solution and the evaluator class was used to store the objectives of the solutions (in this case cost and value of solution). Also the ProblemModule class was used to bind the creator, decoder and evaluator together. Once the implementation is done the project is exported from eclipse into a jar file which is put into the plugins folder of the framework and OPT4J is ran. In the framework interface the selector is selected as NSGA-2 as well as the evolutionary algorithm from the optimizer and the viewer as an output. After that the problem called “GA” is selected and ran. This outputs the results from the algorithm being ran over a certain population. The output is presented by an Archive Monitor which stores all the good solutions. A population monitor which shows the final population, a pareto plot which is a projection of all the good solutions and a convergence plot which shows how the best solution improved over the generations.

Genetic Algorithm

The genetic algorithm is ran from the OPT4J framework interface. It contains 5 different variables. The first one is generations which is the number of generations the algorithm is ran. In this case the generations are preset to a 1000. The alpha value shows population size which is set to 100. The mu

variable is number of parents which is set to 25. The lambda variable is the number of offspring which is set to 25 and the crossoverRate is 0.95 which means that 95% of crossovers will occur. The algorithm is ran using the NSGA-II algorithm for the crossover. The NSGA-II algorithm is preprogramed in the OPT4J framework. This algorithm utilises the distance that an individual holds from the rest of the population and tries to provide a wider Pareto frontier. The way that this works is by selecting the top of the population and passes it over to the next generation, which is the elitist part of the population, then it takes the top half of the population and by crowding distance sorting it creates new solutions which are used to fill up the rest of the new population.

Random Search

The random search algorithm in this project uses the same genotype to create solutions and the two fitness functions from the multi-function objective to evaluate them. Random search is already implemented in the OPT4J Framework and can be used the same way as the Genetic algorithm. There are 2 variables which need to be specified. One of them is the number of iterations which in this case is set to 1000, and the other one is the batchsize which is the number of evaluated solutions which create the pareto frontier. In this case that is set to 25.

Results

Multi-function Objective

In figure 1 below, it can be seen the results from the non-realistic nrp1 data text file. The results show a pareto plot from the archive of the solutions which are considered optimal. The x-axis represents the cost of the solutions and the y-axis is the value of the solutions. As can be seen from the graph the higher the cost means the higher the value. In this case the algorithm has found the best value solutions for the least value costs. Under that in Figure 2, it can be seen a part of the archive monitor which shows the optimal solutions, as well as a list of requirements that is included in those solutions the total cost and the total value of the solutions.

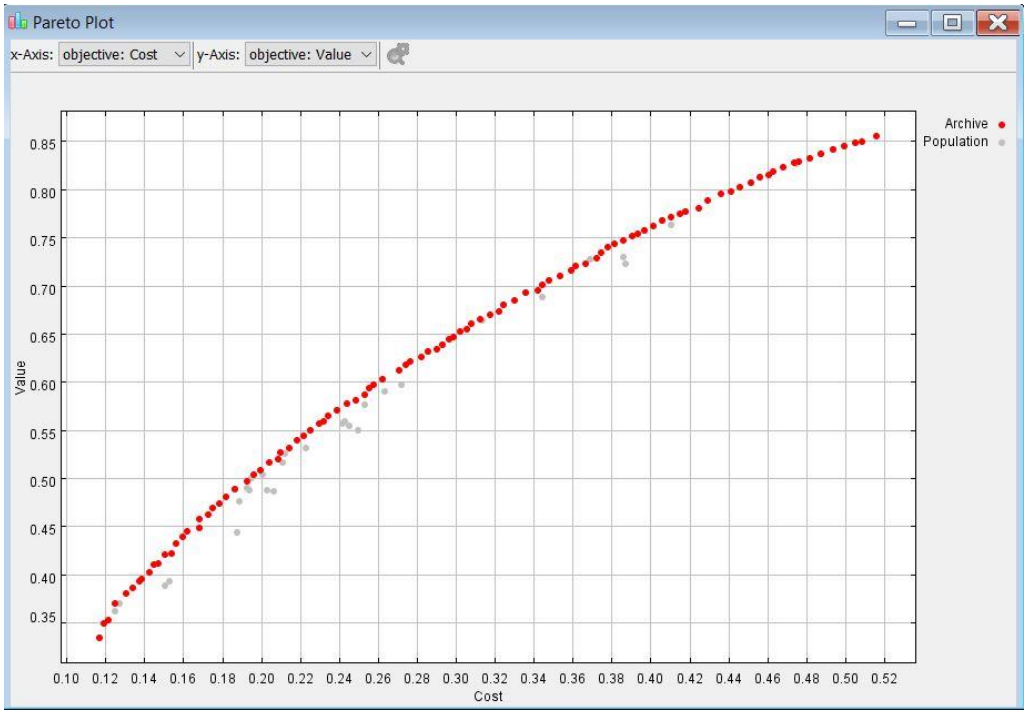

Figure 1: Pareto plot from the non-realistic nrp1.txt file.

Archive Monitor

Size: 100 Auto Update

#	Individual	Cost (MIN)	Value (MAX)
1	[3, 15, 23, 26, 37, 38, 43, 47, 51, 57, 59, 64, 73, 75, 78, 104, 116, 120, 123, 133, 134]	0.11668611435239203	0.33412971238684536
2	[3, 15, 16, 23, 24, 28, 37, 38, 43, 47, 51, 59, 64, 73, 75, 78, 104, 116, 120, 123, 133, 134]	0.11901983663943987	0.34877392001833385
3	[3, 15, 16, 23, 24, 28, 32, 37, 38, 43, 47, 51, 59, 64, 73, 75, 78, 104, 116, 120, 123, 133, 134]	0.1213535589264877	0.35294488369428206
4	[3, 15, 16, 23, 24, 32, 37, 38, 43, 51, 59, 64, 73, 75, 78, 93, 104, 116, 120, 123, 133, 134]	0.12485414235705947	0.3696230090523662
5	[2, 3, 15, 23, 24, 28, 32, 37, 43, 47, 51, 59, 64, 73, 75, 78, 81, 93, 104, 120, 123, 133, 134]	0.13068844807467908	0.3803769909476337
6	[2, 3, 15, 16, 23, 24, 28, 32, 37, 38, 43, 47, 51, 59, 64, 73, 75, 78, 93, 104, 116, 120, 123, 133, 134]	0.13418903150525083	0.3853500630228028
7	[3, 15, 16, 23, 24, 32, 37, 38, 43, 47, 51, 59, 61, 64, 73, 75, 78, 93, 104, 116, 120, 123, 133, 134]	0.1376896149358226	0.3924143462816546
8	[2, 3, 15, 23, 24, 26, 32, 37, 38, 43, 47, 51, 57, 59, 64, 73, 75, 78, 93, 104, 116, 120, 123, 133, 134]	0.1388564760793465	0.39518162025896636
9	[2, 3, 15, 16, 23, 24, 32, 37, 38, 43, 51, 57, 59, 64, 73, 75, 78, 81, 91, 93, 104, 120, 123, 133, 134]	0.14235705950991828	0.40208548183797405
10	[2, 3, 15, 16, 23, 24, 32, 37, 38, 43, 51, 59, 64, 73, 75, 77, 78, 81, 93, 104, 116, 120, 123, 133, 134]	0.14469078179696612	0.41001489629884263

Figure 2: Archive Monitor from the non-realistic nrp1.txt file.

Figure 3 and 4 below show the pareto front and a part of the archive monitor but for the realistic nrp-g1.txt data file. The archive monitor is provided to show how the solutions are represented and the values they hold.

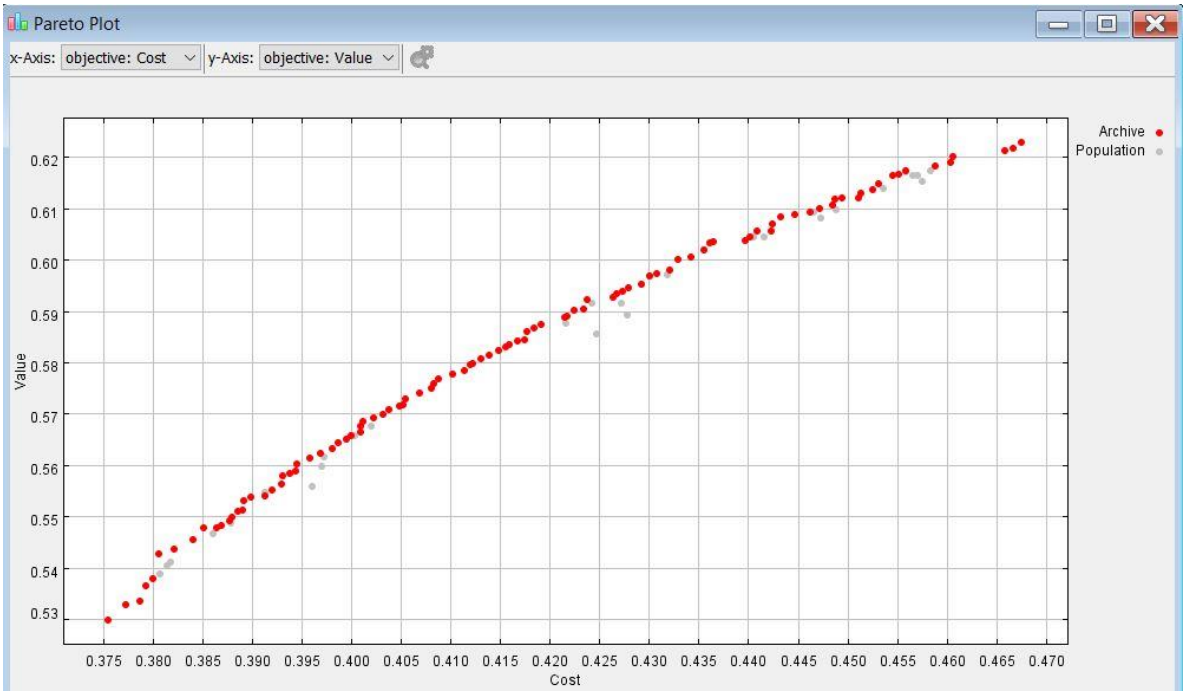

Figure 3: Pareto plot from the realistic nrp-g1.txt file.

Population Monitor

#	Individual	State	Cost (MIN)	Value (MAX)
1	**[4, 10, 11, 15, 16, 20, 21, 24, 25, 28, 32, 33, 34, 35, 36, 37, 38, 40, 41, 43, 47, 49, 54, 56, 57, 59, 60, 61, 6...**	**evaluated**	**0.4675001882955427**	**0.6229296129844439**
2	**[10, 11, 15, 16, 20, 21, 24, 28, 34, 36, 37, 40, 41, 43, 47, 49, 54, 56, 59, 60, 61, 62, 64, 65, 66, 67, 71, 74, ...**	**evaluated**	**0.3773442795812272**	**0.5327240321699963**
3	**[3, 4, 6, 10, 11, 15, 16, 17, 20, 21, 24, 25, 28, 33, 34, 35, 36, 37, 38, 40, 41, 43, 47, 49, 54, 56, 57, 59, 60, ...**	**evaluated**	**0.46584318746704223**	**0.6213132098730614**
4	[10, 11, 15, 16, 20, 21, 24, 28, 34, 36, 37, 38, 40, 41, 43, 47, 49, 54, 56, 59, 60, 61, 62, 64, 65, 66, 67, 71, 74, 76, 79,...	evaluated	0.380733599457705	0.53881220722537
5	**[1, 10, 11, 15, 16, 20, 21, 24, 28, 31, 34, 36, 37, 38, 40, 41, 43, 47, 49, 54, 56, 59, 60, 61, 62, 64, 65, 66, 6...**	**evaluated**	**0.3886420125028206**	**0.5510316282541281**
6	**[10, 11, 15, 16, 20, 21, 24, 28, 34, 36, 37, 38, 40, 41, 43, 47, 49, 54, 56, 59, 60, 61, 62, 64, 65, 66, 67, 71, ...**	**evaluated**	**0.3787753257512955**	**0.5333427966724197**
7	**[3, 4, 10, 15, 16, 20, 21, 24, 27, 28, 31, 33, 34, 35, 36, 37, 38, 40, 41, 43, 47, 49, 54, 56, 57, 59, 60, 61, 62,...**	**evaluated**	**0.4462604504029469**	**0.6092932914072505**
8	**[10, 11, 15, 16, 20, 21, 24, 28, 34, 36, 37, 38, 40, 41, 43, 47, 49, 54, 56, 59, 60, 61, 62, 64, 65, 66, 67, 69, ...**	**evaluated**	**0.386909693454843**	**0.5481885455764994**
9	**[4, 6, 10, 11, 15, 16, 20, 21, 24, 28, 31, 33, 34, 36, 37, 38, 40, 41, 43, 47, 49, 54, 56, 57, 59, 60, 61, 62, 64,...**	**evaluated**	**0.4233637116818507**	**0.5903662100502938**
10	[3, 4, 6, 10, 11, 15, 16, 20, 21, 24, 27, 28, 31, 33, 34, 35, 36, 37, 40, 41, 42, 43, 47, 49, 54, 56, 57, 59, 60, 61, 62, 64,...	evaluated	0.458311365519313	0.6172454437883774

Figure 4: Archive Monitor from the realistic nrp-g1.txt file.

Single-function Objective

Figure 5 below shows a convergence plot when applied the genetic algorithm with a single objective function to the non-realistic nrp1.txt data text file. In order for the single function to provide a valid solution a maximum cost needs to be input. The figure below is for a maximum cost of 500, when the total cost of all requirements is 2909. The single function objective in this case comprises of 90% of the total value of the solution and 10% of the cost of the solution. The convergence plot shows how the solutions have improved throughout the 1000 generations.

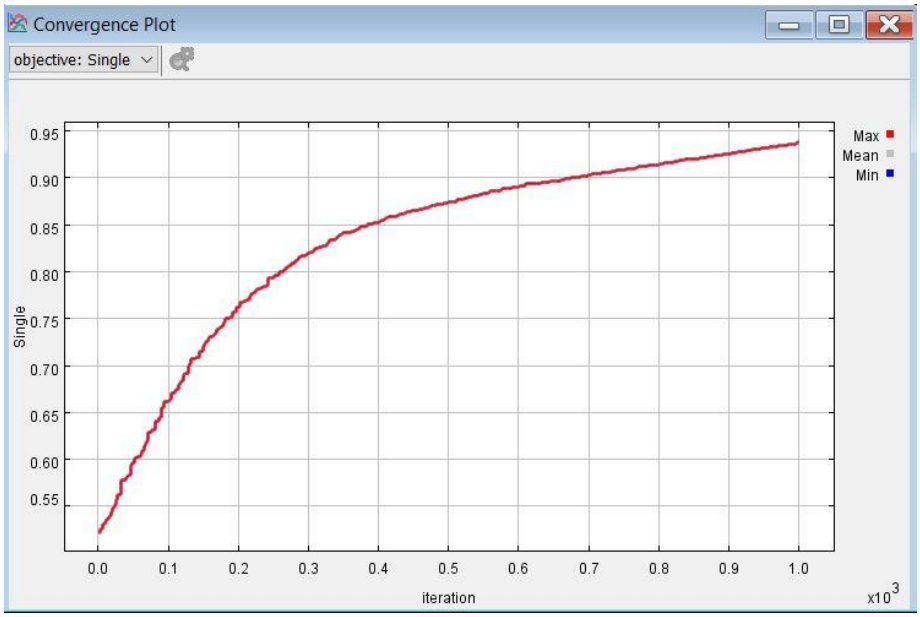

Figure 5. Convergence plot from the non-realistic nrp1.txt file

Figure 6 below shows a convergence plot for a Single objective genetic algorithm used on the realistic nrp-g1.txt data file. In this case the total cost of all the requirements is 13227 and the solution found is for a maximum cost of 2000. The single objective function is again calculated by taking into consideration 90% the value of the solution and 10% the cost.

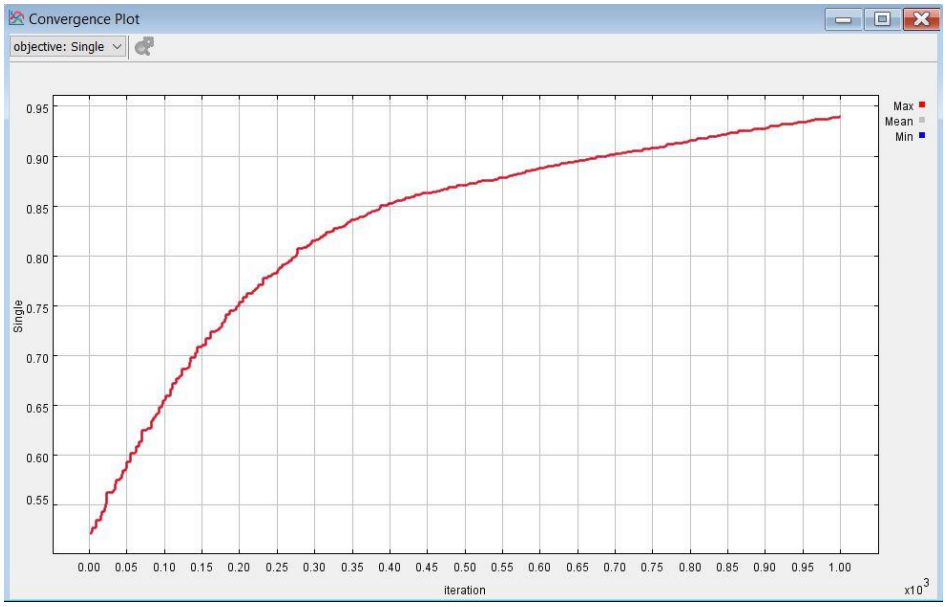

Figure 6: Convergence plot from the realistic nrp-g1.txt file.

Random Search

Figures 7 and 8 below show the Pareto plot for the Multi-Objective Random Search with 1000 iterations. The first one is for the non-realistic nrp1.txt file and the second is from the realistic nrp-g1.txt file

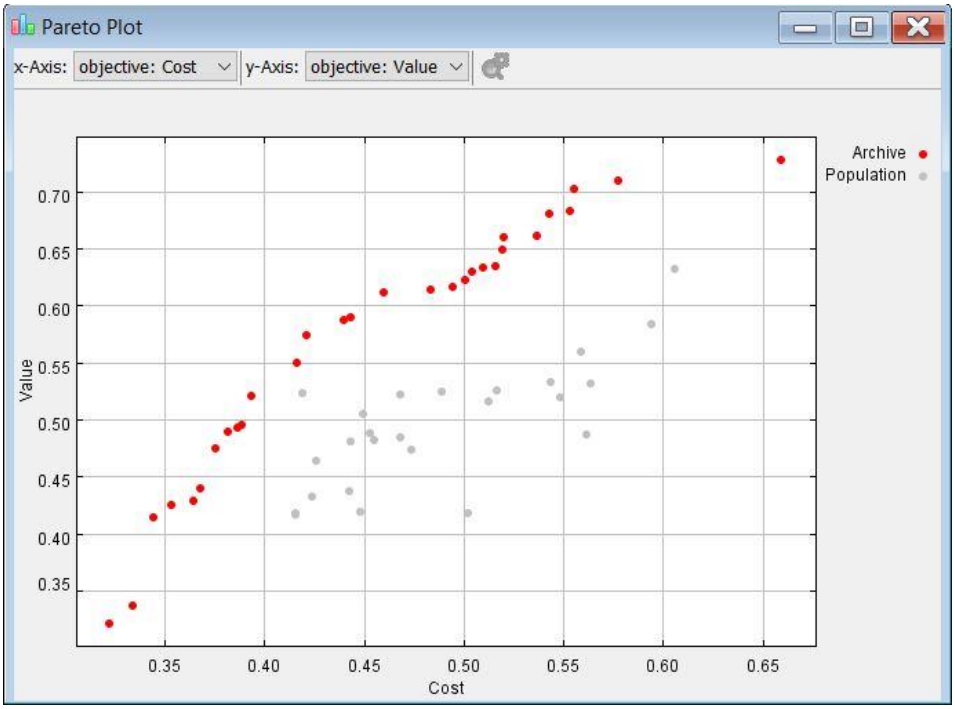

Figure 7: Pareto plot for Multi-Objective Random Search from the non-realistic nrp1.txt file.

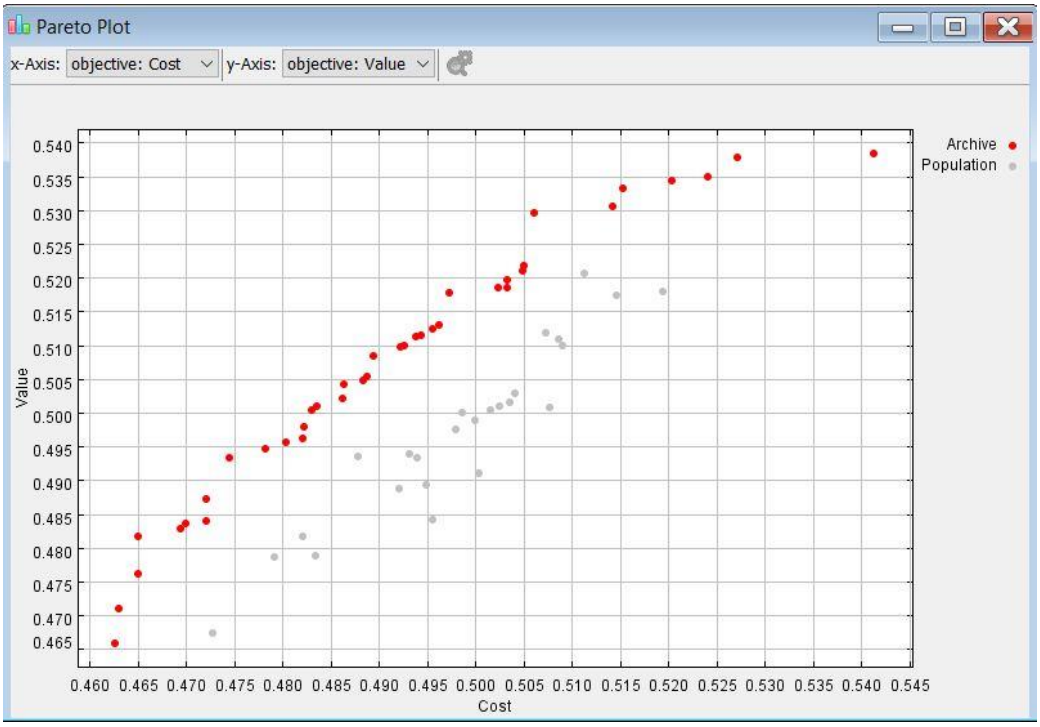

Figure 8: Pareto plot for Multi-Objective Random Search from the realistic nrp-g1.txt file.

Figures 9 and 10 below show the Convergence plot for the Single-Objective Random Search with a 1000 iterations. The first one is from the non-realistic nrp1.txt file and the second is from the realistic nrp-g1.txt file.

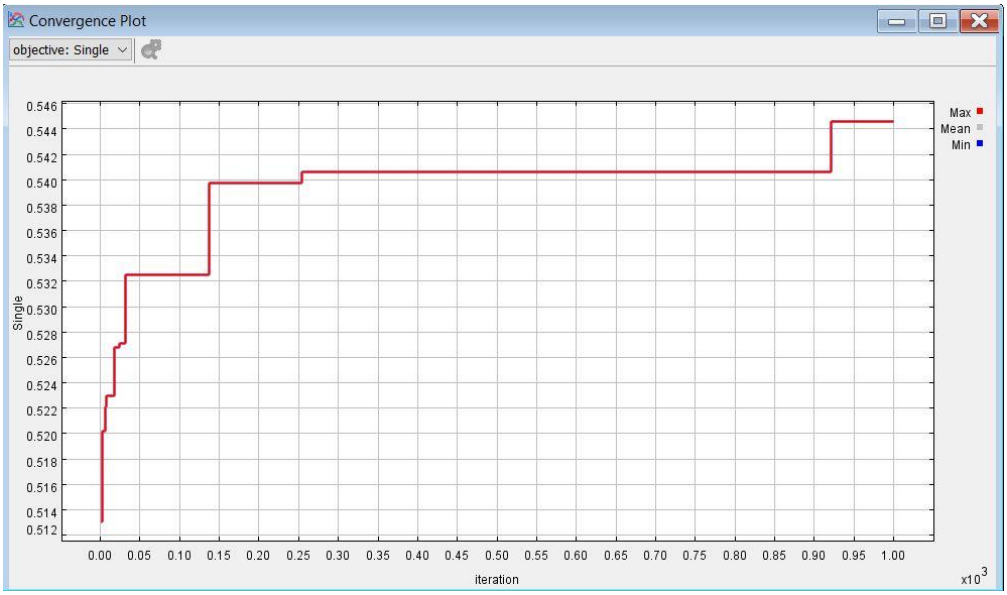

Figure 9: Convergence plot for Single-Objective Random Search from non-realistic nrp1.txt file.

Figure 10: Convergence plot for Single-Objective Random Search from realistic nrp-g1.txt file.

Comparison between GA Multi-Function Objective and Single-Function Objective and Random Search

The figures provided above can be used to make a comparison between how well the Multi-Function Objective and Single-Function Objective work for the Genetic Algorithm and also compare both of them with their respective results from the Random Search Algorithm. The main difference between

Multi-Function and Single-Function Objectives is that the Multi-Function Objective provides a set of solutions which show the best solution for different costs, while Single-Function Objectives only provides 1 solution for a predetermined maximum cost of that solution. This gives the option to the user to pick the best suited solution without having to set a maximum cost first. Also from the results from the genetic algorithm and the random search algorithm it can be seen that the genetic algorithm has a more targeted approach and is going to reach better solution every time it is ran. To compare the data from the Multi-Function Objective for Genetic Algorithm and Random Search Algorithm. The Genetic Algorithm provides a more consistent and wider Pareto frontier and also gives better value results for lesser costs than the Random Search. As for Single-Function Objective, the best solution from the Genetic Algorithm is almost twice better than the Random Search even though they have the same maximum cost set and the same weight between cost and value.

How to run

In order to run the .jar file submitted it needs to be loaded onto eclipse where in the GACreator class the full directory and the filename of the desired .txt file to run needs to be copied in. After that it needs to be exported again and put into the plugins folder and loaded onto the framework. The submitted jar file is pre-set to run the Multi-Function Objective. In order to change to Single-Function in the GAEvaluator class lines 17 and 32 through to 38 need to be uncommented and lines 27 and 28 need to be commented.