Combining Multi-objective Evolutionary Algorithms with Machine Learning Algorithms to Solve Real World Engineering Problems

Abstract

Multi-objective Optimisation methodologies are used to find solutions to several types of problems that are tackled in science. This paper focuses on solving engineering problems that have several objectives, whereby these objectives are conflicting one another. Scientist have approached these problems through several proposed methodologies that require multi-objective algorithms to find a solution set. However, finding an optimal solution can be challenging due to the solution set being large the optimum value is difficult to find. To approach the tremendous complexity of real-world problems several methodologies are being used in hopes of find a true solution. This study is about the combination of two different types of algorithms that work together to find the solutions to sixteen real world problems. For this study the chosen multi-objective algorithm is the NSGA II (Non-dominated Sorting Genetic Algorithm) combined with the Neural Network machine learning models to find solutions to the given problems. The combination of the two algorithms is implemented in Python with the use of SKLEARN and PYMOO which are libraries that are used to merge the functionalities of the two types of algorithms. Results of the combination are evaluated through the final product which is the Hypervolume of the NSAG2 and the RMSE and R2 of the Neural Networks and other methods of analysis that shall be used to analyse the performance of the methodology proposed. The quality of the performance of the combination determines which solution sets are of the highest conditions, where the optimum value is located.

Introduction

Engineering problems are characterized of conflicting objectives which requires the use of multi-objective optimisation. However, most of the problems that are solve by the MOEAs are synthetic due to the complexity characterised by the real-world problems. The problem suite that are being used in this study is provided by . The problem suite is implemented in Python. A re-implementation given by [ ] can be access through a GitHub repository. The re-implementation of the problem suite in written in an object-oriented approach where each problem is viewed as an object. To implement the combination of proposed methodology this required the use of PYMOO a library that can be used in a Python environment to express the chosen MOEA. The problem suite re-implementation in utilised in accessing attributes for each problem which included the number of variables, number of objectives and the lower and upper bounds of the features and the original objective function of the

problem. The re-implementation will not be used in the experimental stage of this paper due to requirements of the use of Pymoo to define these problems differently. Therefor the problem objects have the same attributes and functions however are expressed differently in the proposed methodology implementation.
Machine Learning is widely mentioned in this paper. To implement the proposed methodology of this paper, SkLearn which provide several machine learning algorithms is used to build the Neural Networks models that are used to predict the objective values for a given set of variable values, this topic is discussed in depth in section .Theoretically, neural networks are used in Artificial Intelligence to predict numbers and information when given a set of data. Several variants of neural networks predict different types of data, and also different types of prediction or forecasting. The background of machine learning and several topics that revolve around Neural Networks is discussed in section .

To determine the performance of the algorithm several methodologies can be used to assess which solution set is of the highest quality. For the proposed methodology the Hypervolume of the NSGA2 is used to determine the performance in finding the solution set for each problem. For the neural networks on the other-hand RMSE (root mean squared error) and the R2 (R squared) values are used to determine the quality of the predictive models used as surrogates in the objective function. These components are discussed along with the experimental information gained during the approach of the proposed methodology and is discussed in depth in section [.] .

Technical information surrounding the methodology used to implement the combination for the two algorithms are discussed in section

of Determination (R2)

2 Real World Problems

Real-world problems have numerous conflicting objectives, which makes them hard to solve in real life. Similarly in this study, in this study, they must be expressed as multi-objective problems so that their attributes are able to be quantified and implemented for multi-objective evolutionary algorithms to interpret. As mentioned before the real-world problems are presented as a problem suite which is accessed through GitHub [2].

Since this methodology has already been discovered [1] , it was used to solve synthetic test problem or test functions because of the complexity of real world problems, feasible solutions are hard find. Multi-objective Evolutionary algorithms provide the objective space for a given problem. In multi-objective optimisation, the chosen algorithm can initiate a population and these solution population evolves to approximate the Pareto optimal set, which is represented by the objective space. Populations that survive the algorithm could be large resulting to a large objective space which is composed of uncountable number of solutions to a single problem.

The numerous numbers of existing solutions given by the objective space doesn't remain as the only obstacle in solving real world problems. The absence of constraints in real world problems is another drawback into finding a set of feasible solutions. Constraints are conditions that must all be satisfied for the Multiobjective Evolutionary Algorithm to declare the solution valid.

In the GitHub re-implementation, there are no constraints found for the problems that are to be used for this study.

3 Multi Objective Evolutionary Algorithms

This section is about the theoretical background surrounding Multiobjective Evolutionary Algorithms. MOEAs are used to solve MOPs. In this implementation, the machine learning algorithms are used as metamodels to represent as surrogates for the real objective function.

9 Pymoo

Pymoo is used to implement the NSGA2 algorithm in python. This section is about the utilisation of the NSGA2 and the technical information in how it was used for the proposed methodology combination.

The NSGA2 is accessed in python through the pymoo library. For the combination implementation. The problem had to be defined using the *ElementwiseProblem* class. This enables the NSGA2 to interpret the engineering problems.

The problem is initiated by calling the minimize function which combines the chosen algorithm, NSGA2 and the problem object which is the instantiation of the Problem class of type *ElementwiseProblem.* The NSAG2 accesses the problem class which can be one of the problems in the problem suite and uses the evaluation function to retrieve objective values from a randomised variable values corresponding to the upper and lower bound attributes of the problem.

In this study, the implementation used requires the neural network models to be called inside the evaluate function that the NSGA2 utilises to generate a population of solutions. As the NSGA2 algorithm converges, a set of variable and objective values can be retrieved after termination. These values are used to train the neural networks again until the number of cycles in satisfied.

10 SKLEARN

SKLEARN is used to implement the neural networks int the proposed methodology. The type of neural networks that is used for the proposed methodology is multilayer perceptron regression, which is a type of regression analysis mentioned in section 4.

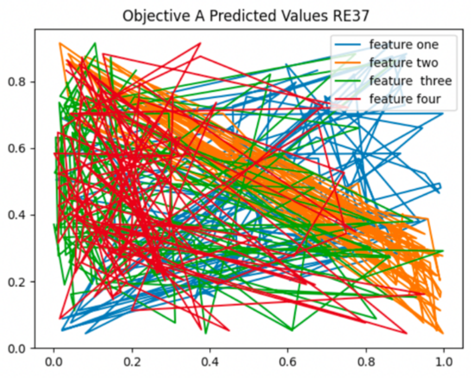
During implementation the neural networks act as a surrogate or metamodels that are used to predict the values for each objective. Such models must be trained by a data set is initially randomised at the beginning. The neural networks must be built before the NSGA2 is called for each iteration.

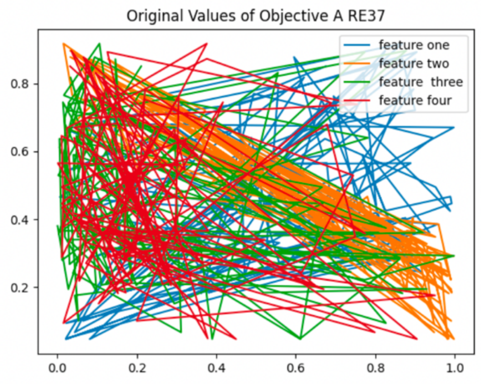
For this implementation of the combination of NSGA2 and neural networks, the neural networks must be trained by a new dataset each time. Pymoo only allows the neural networks to train once, therefore the neural networks are built for each iteration by training them with a larger dataset for iteration building a stronger model that is able to predict an accurate value during their call inside the evaluate function located in the problem class

Testing and Experimental

For the Neural Networks to be build data must use to train the models to enable prediction. Several approaches were done to retrieve the data that are used to build the models. Since the variable values have to be trained to the neural networks utilises the lower and upper bounds the NSGA2 is able to randomise the values that are within the range that satisfies the problem characteristics. However, the objective values from the NSAG2 are true objective values and not randomize, neural networks can able to predict the objective values easily which implies less training and smaller size of data is used to build them and they can predict a value that is close to the true value of the objectives.

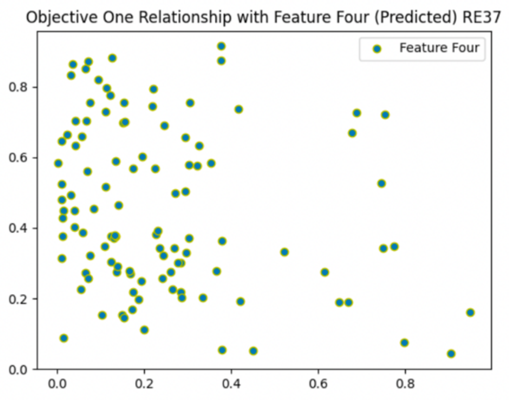
This section mainly starts with the experimentation done with the neural networks. Neural networks can predict the values of a given data set. By utilising sklearn I have managed to view relationships between the objectives and the features. The diagrams below shows the results of the neural networks with the use of data from the NSGA2 algorithm to retrieve the variable values and objective values.

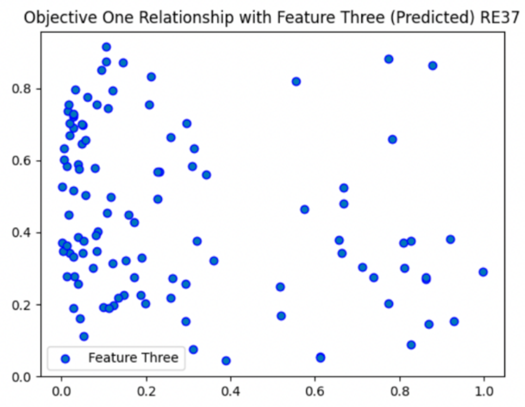




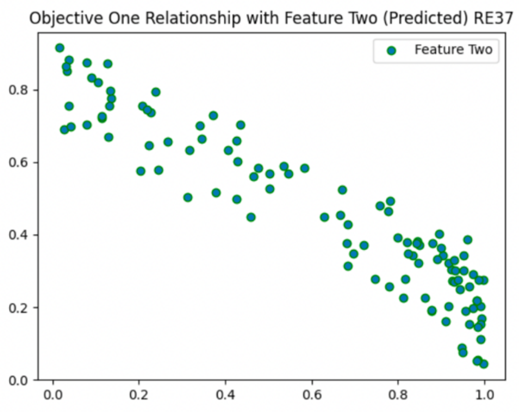
Diagrams [1] and [2] clearly shows the predicted values and the true values are very similar. Moreover, when viewed very closely in the PyCharm software, the predicted values are less scattered compared to the original values.

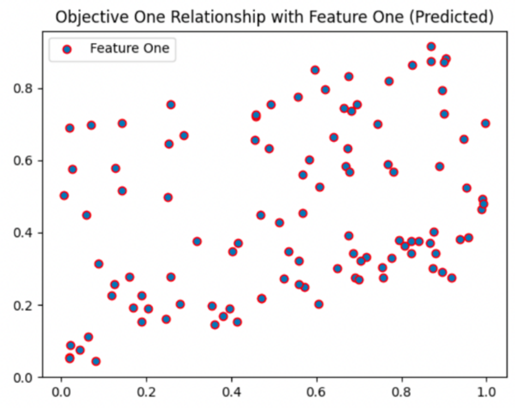
The diagrams below show the relationships of the predicted objective function values and the features





Diagrams below, [3] and [4] that shows the relationships between the predicted values of objective one with four and three





Diagrams [5] and [6] that shows the relationship between the predicted values of objective one with features two and one.

Several approaches were done to combine NSGA2 and Neural Networks. Complications due to the availability of functions provided by Pymoo and Sklearn resulted to different approaches to the implementation. This section is about the experimental stage of the combining the NSGA2 and the Neural Networks.

For the code, the pymoo library allowed access to several multi-objective algorithms. As mentioned before the Non-dominated Sorting Algorithm II (NSGA2) is a multi-objective evolutionary algorithm that is inspired by genetic algorithms. Furthermore, pymoo also offered some other types of algorithms other than multi-objective algorithms such as single-objective and many objective algorithms. The pymoo library also allows the programmer to utilise built in classes that can be used to define the several problems.

Several methodologies were available in defining the problem choice namely.
The NSGA2 implementation that pymoo provides requires the programmer to define the problem in a specific format where the characteristics are presented in a slightly altered method from the original re-implementation [paper providing the problems]. For the NSGA2 algorithm to comprehend the problem characteristics, it has to be defined as an object with metadata. The metadata for a problem includes the number of variables, number of objectives, number of constraints and the lower and upper bounds.

The several methodologies that may be used in defining a problem includes P*roblem*(vectorized), *ElementwiseProblem* (loop), *FunctionalProblem(*loop), and *Adding a known optima.*

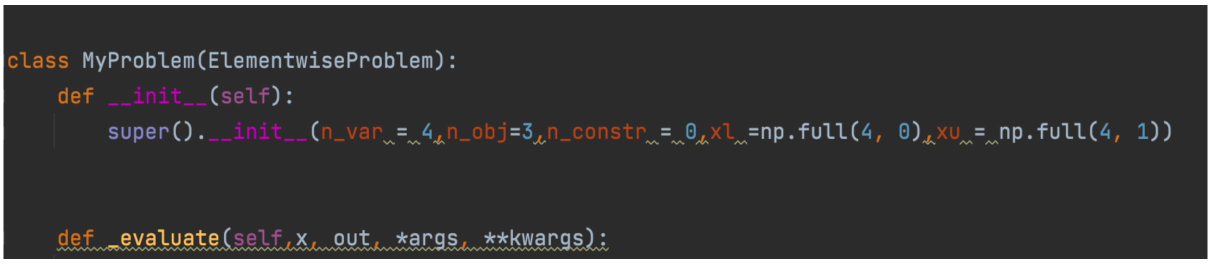
At the beginning of the experimentation stages where numerous information is unknown. I have tried defining the problem in each possible method that pymoo allows. This includes implementing the problem using the Problem class provided. The Elementwise Problem is the only way of defining the real-world problems. It works by implementing the problem as an object-oriented definition. The evaluate function must be written inside the problem class. The Elementwise Problem implementation requires the lower and upper bounds (xl and xu) to be written in a vectorised format.

During at which the NSGA2 utilises the problem object defined by the user. The NSGA2 or any algorithm looks for the evaluation function of the problem object and randomise a set of variables that satisfy the lower and upper bound characteristic of the problem object. For Elementwise problem object. The evaluate the function retrieves a vector of length 10, and it is called for each solution.

The Functional Problem on the other hand required me to write a lambda function that represents the objectives and constraints for the problem. However, it is stated that many calls have to be made to retrieve a solution set for a given set of variable values.

Another method proposed by pymoo is by adding a known optimum. This requires the user to create a function that calculates the pareto front of the problem, if the optimum is not known. This methodology was approach however, adding a known optima is difficult if the problems don't have any known constraints.

The diagram below shows the implementation of a single problem using the element wise problem class provided by pymoo.



```
class MyProblem(ElementwiseProblem):
    def __init__(self):
        super().__init__(n_var = 4,n_obj=3,n_constr = 0,xl =np.full(4, 0),xu = np.full(4, 1))


    def _evaluate(self,x, out, *args, **kwargs):
```

Diagram [ ] : Problem definition for the real life problems are in this format for the NSGA2 to deploy. The metadata are *n_var, n_obj, n_constr, xl and xu.* The metadata is used by the NSGA2 to generate the randomised variable values which corresponds to the lower and upper bounds given by the user to retrieve a set of variable values that are passed through the real objective function of the problem. However, the combination of NSGA2 and Neural Networks requires the neural networks prediction to be utilised for the NSGA2.

During the experimentation of combining the NSGA2 and the Neural Networks, a lot of methodologies were used to approach to retrieve the proposed results. However, there were a lot of complication during the end of the stage. One of the complications were that *sklearn* doesn't allow the neural networks to be trained again. Therefore, during the experimentation, the neural networks had to be trained by a larger set of data and are declared again inside a combination loop. The combination loop represents the cycles entered by the user. The user enters a number N and it represents how many iterations the combination of the neural networks and the NSGA2. An assumption would be that if the archive is increased for each iteration which contains the original function values and the corresponding variable values, the neural network models are able to predict a much more accurate value that represents the objective values the NSGA2 uses to find solutions.

The archives which hold all the values that are used to train the model declared outside the combination loop, and functions inside the loop can modify the archives to update the values and operations are able to access again to train the new neural networks that are again by the algorithm.

The combination function holds all the operations that run when the two algorithms are combined.

The steps needed before the combination begins are:

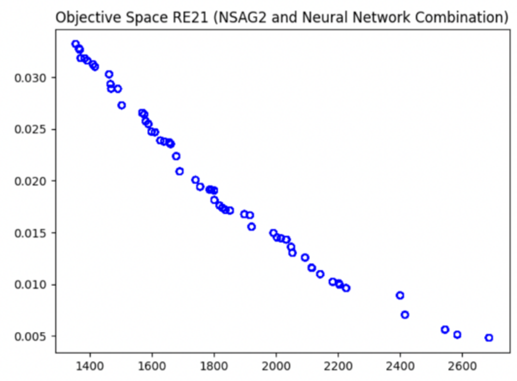
- The neural networks must have been defined and trained at the very beginning. This requires a data set to train, and the datasets must satisfy the lower and upper bound of the problem features.

- The neural networks must be called inside the problem class located in the evaluation function which predicts a value when the NSGA2 initializes a set of random variable features for the function. And the neural networks must return a value representing the objective function values for the given variable set.
- Each objective value is represented by the corresponding neural network that predicts that specific objective. Which means that for each neural network, they must have been trained by the specific objective values that correspond to the variable feature values.
- The neural networks are predicting a value for a given set of variable values. The neural networks must be of a regression type model (In the experimentation, all the neural networks are of the type Multilayer Perceptron Regression model).

The steps that occur during combination are:

- The NSGA2 function is initiated. For algorithms that are provided by pymoo it is initiated by the minimize function, which returns a set of objective values and variable values.
- The set of variable values are passed through the original objective function, which returns a set of objective values that are true for each corresponding variable set.
- The previous step indicates that there are more objective values. And for the neural networks to train, the number values of X (variable) and Y (objective) must be equal otherwise errors would occur during training. To justify the previous statement regarding the difference in size of the datasets. For initialization both have the same number of values (the dataset used to build the neural networks at the very beginning). Another dataset is retrieved after the NSGA2 performs, which is another dataset of X and Y. Obtaining the objective values from the original function, which is the same values that were provided by the NSGA2 algorithm, means that the objective values exceed by a certain amount compared to the variable values.
- To avoid errors during the neural network training, the variable values from the NSGA2 is double in the variable archive.
- As mentioned before the archive is located outside the combination loop. In this step the archives are accessed again to build the new neural networks. Operations are used to prepare the datasets for the neural networks. And a new set of neural networks are trained again.

The diagrams below shows the results with the NSAG2 and the Combination of the NSGA2 and the Neural Networks



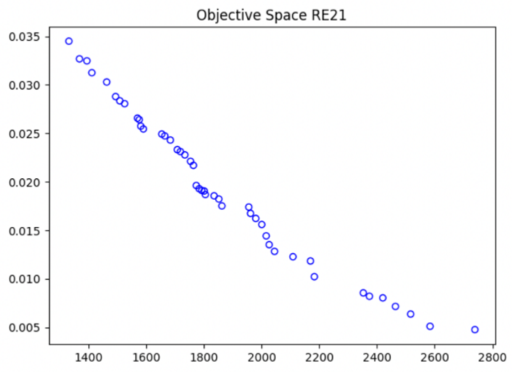
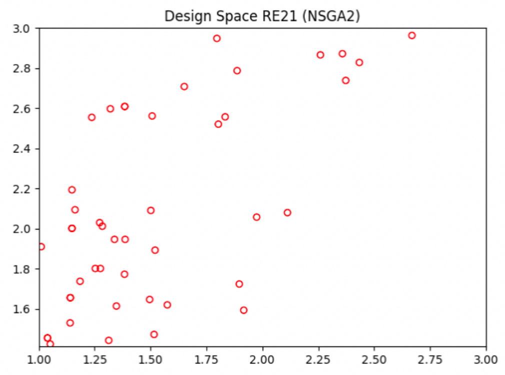


Diagram [7] and [8] shows the Combination of NSAG2 and Neural Networks and when only using the NSAG2.

Objective space of the combination shows a more aligned set of results representing the objective values of the variables. The combination algorithm performed more than the NSAG2 alone. However, if the NSGA2 algorithm alone performed a large number of iterations, the alignment could also be achieved.

The diagrams below show the Design Space (using the variable values of RE21 problem)



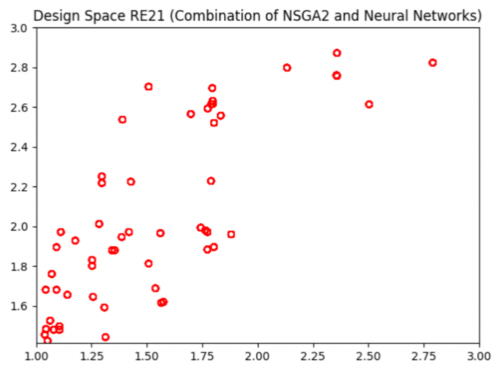


Diagram [9] and [10] showing the design space of the RE21 problem with and without Neural Networks. The combination shows slightly less scattered points compared to the NSGA2 by itself.

As seen in the previous diagrams the objective space for problems such as RE21 are seen in two dimensions. For several problems that exceed two objectives, the objective space is visualised through 3D scatter plots and for more objectives they can be interpreted in Parallel Coordinate Plots.

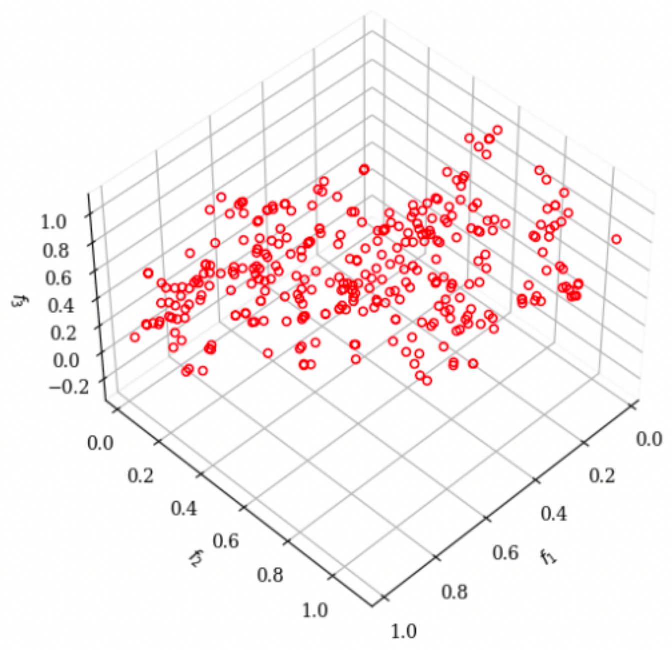


Diagram [11] shows the objective space for the RE37 problem in 3 dimensions.

Parallel Coordinate Plots are used to plot problems with objectives more than three. The diagram below shows the parallel coordinate plots for problem RE31.

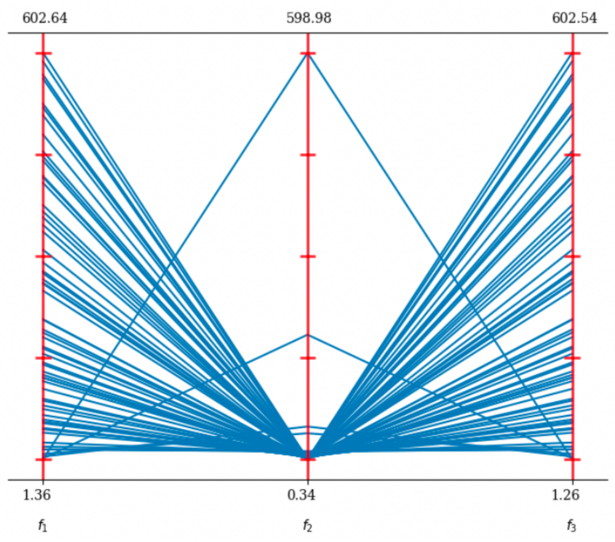


Diagram 12: Parallel coordinate plots

Product and Fitness for Evaluation

As stated in the project specification, the hypervolume is the desired product. The Hypervolume represents the performance of the combination implementation. However, we couldn't retrieve the hypervolume of the problems because of the NSGA2 was not able to find a pareto front for any of the problems.

The pymoo library didn't allow for a hypervolume to be found because the problems are interpreted as custom data. The library only allows test problems that are provided by pymoo are the problems that can be used which has a known pareto front. However, Pymoo offers several methods to analyse the convergence of the algorithms. For each problem run through an algorithm, during termination, there are two possible scenarios that has to be determined. One scenario is when the Pareto-front is not known and the when the. Pareto-front has been derived analytically or an approximation has been made to represent the pareto front. Several methods are used to approximate the pareto front, as mentioned before the programmer can inject their own calculation to calculate the pareto front by adding a known optimum to the problem class. However, in the case of real world problem where the pareto front is not known there are several ways analyse how the algorithm performed.

To determine the performance of the algorithm or the solution set a running metric has to be used to show how to objective space changes between different generations and uses the algorithms survival to visualize improvement. The minimize function that is used to start the algorithm contains the history attribute that analyses the difference of the objective space for each generation.

The diagram shows the running metric of the combination algorithm used for RE21

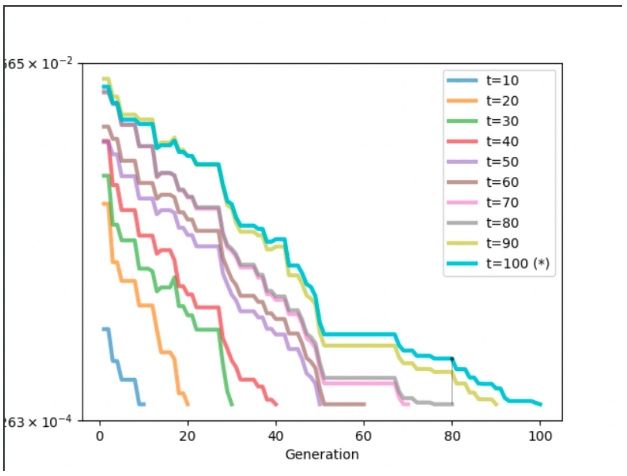


Figure [13] to show the metric for the algorithm used with the neural networks prediction in finding the solution set for RE21.

Metrics are used to determine the performance of Multi-objective Evolutionary Algorithms . To accomplish a fair comparison on different solutions sets retrieved from the algorithm, metrics are used to analyse the objective space between each generation the algorithm performs. Performance metrics are used to measure shows the performance of the algorithm throughout the run, which analyses if the algorithm was consistently improving from the start until termination or were there sudden spurts of improvements during the performance. Running metrics are used to profile the algorithm's generation performance.

At the end of the experimental stage, I have found out that the code performs very slowly when applied to problems with more objectives. The errors occur when the operations inside the loop are accessing the archive and its values to train the neural networks. The Archive causes a lot of delays and only works on problems with the fewest number of objectives.

8 References

[1] Chugh, T., Sindhya, K., Hakanen, J., Miettinen, K. (2019). A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms. Soft Computing, 23(9), 3137-3166.

[2] Tanabe, R., Ishibuchi, H. (2020). An easy-to-use real-world multi- objective optimization problem suite. Applied Soft Computing, 89, 106078.

[3] Coello, C. A. C., Lamont, G. B., Van Veldhuizen, D. A. (2007). Evo- lutionary algorithms for solving multi-objective problems (Vol. 5, pp. 79-104). New York: Springer.

[4] Pham, D. T., Ghanbarzadeh, A. (2007, July). Multi-objective optimi- sation using the bees algorithm. In 3rd International Virtual Conference on Intelligent Production Machines and Systems (p. 6).

[5] Zhou, A., Qu, B. Y., Li, H., Zhao, S. Z., Suganthan, P. N., Zhang, Q. (2011). Multiobjective evolutionary algorithms: A survey of the state of the art. Swarm and evolutionary computation, 1(1), 32-49.

[6] Cagnina, L. C., Esquivel, S. C., Coello, C. A. C. (2008). Solving en- gineering optimization problems with the simple constrained particle swarm optimizer. Informatica, 32(3).

[7] Nadimi-Shahraki, M. H., Taghian, S., Mirjalili, S. (2021). An improved grey wolf optimizer for solving engineering problems. Expert Systems with Applications, 166, 113917.

[8] Mazhoud, I., Hadj-Hamou, K., Bigeon, J., Joyeux, P. (2013). Particle swarm optimization for solving engineering problems: a new constraint-handling mechanism. Engineering Applications of Artificial Intelligence, 26(4), 1263- 1273.

[9] Adeyemo, J. A., Otieno, F. A. O. (2009). Multi-objective differential evo- lution algorithm for solving engineering problems. Journal of Applied Sciences, 9(20), 3652-3661.

[10] Sattar, D., Salim, R. (2021). A smart metaheuristic algorithm for solving engineering problems. Engineering with Computers, 37(3), 2389-2417.

[11] Jaberipour, M., Khorram, E. (2010). Two improved harmony search algorithms for solving engineering optimization problems. Communications in Nonlinear Science and Numerical Simulation, 15(11), 3316-3331.

[12] Lwin, K., Qu, R., Kendall, G. (2014). A learning-guided multi-objective evolutionary algorithm for constrained portfolio optimization. Applied Soft Computing, 24, 757-772.

[13] Williams, Z. D., Kapfhammer, G. M. (2010, July). Using synthetic test suites to empirically compare search-based and greedy prioritizers. In Pro- ceedings of the 12th annual conference companion on Genetic and evolutionary computation (pp. 2119-2120).

[14] Wright, J., Jordanov, I. (2017). Quantum inspired evolutionary al- gorithms with improved rotation gates for real-coded synthetic and real world optimization problems. Integrated Computer-Aided Engineering, 24(3), 20
[15] Khoroshiltseva, M., Slanzi, D., Poli, I. (2016). A Pareto-based multi-objective optimization algorithm to design energy-efficient shading devices. Applied en- ergy, 184, 1400-1410.

[16] Miglierina, E., Molho, E. (2002). Scalarization and stability in vector optimization. Journal of Optimization Theory and Applications, 114(3), 657- 670.

[17] Li, M., Yang, S., Liu, X. (2015). Pareto or non-Pareto: Bi-criterion evolution in multiobjective optimization. IEEE Transactions on Evolutionary Computation, 20(5),

[18] Multiobjective Evolutionary Algorithms Research: A History and Anal- ysis (Section 2.3) [19] On Measuring Multiobjective Evolutionary Algorithm Performance [20] Multiobjective Evolutionary Algorithms: A Survey of the State of the art [21] Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach [22] Comparison of Multiobjec- tive Evolutionary Algorithms: Empirical Results [23] Applications of Multiob- jective Evolutionary Algorithms [24] A New Optimisation Algorithm to Solve Multi-objective Problems [25] Multi-objective Simulated Annealing for Hyper- parameter optimisation in Convolutional Neural Networks

[26] Beume, N., Fonseca, C. M., Lopez-Ibanez, M., Paquete, L., Vahrenhold, J. (2009). On the complexity of computing the hypervolume indicator. IEEE Transactions on Evolutionary Computation, 13(5), 1075-1082.

[27] Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., Da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on evolutionary computation, 7(2), 117-132.

[28] Guerreiro, A. P., Fonseca, C. M., Paquete, L. (2020). The hypervolume indicator: Problems and algorithms. arXiv preprint arXiv:2005.00515.

[29] While, L., Hingston, P., Barone, L., Huband, S. (2006). A faster algo- rithm for calculating hypervolume. IEEE transactions on evolutionary compu- tation, 10(1), 29-38.

[30] XiuLing, Z., ChengYi, S., Ning, M., WenJuan, L. (2007, September). Generalization of HSO algorithm for computing hypervolume for multiobjective optimization problems. In 2007 IEEE Congress on Evolutionary Computation (pp. 3114-3118). IEEE.

[31] On Using the Hypervolume indicator to compare Pareto Fronts: Appli- cation to Multi-criteria optimal experimental design

[32] Evolutionary Computation and Convergence to a Pareto front [33] Pareto Exploration with Uncertain Objectives [34] Adaptive Weighted-sum method for bi-objective optimization: Pareto front generation [35] Pareto Front Estimation for Decision Making [36] Adaptive Weighted Sum Method for Multi-objective optimization: a new method for Pareto front generation [37] Pareto-front ex- ploitation in symbolic regression [38] Multiobjective Evolutionary Algorithm with controllable focus on the knees of Pareto front