University of Stirling

Faculty of Natural Sciences
Division of Computing Science and Mathematics

Large Scale Global Optimisation of Housing Improvements

Developing Metaheuristics-based Multi-Objective Evolutionary Algorithm (MOEA) driven Software in jMetalPy for Multi-Variable Automated Decision Making

Author:
John Flavian Jabbo

Supervisor:
Dr Alexander Brownlee

Dissertation submitted in partial fulfilment for the degree of Master of Science in Artificial Intelligence

October 2021

Abstract

Many real world optimisation problems have goals that conflict, e.g.; cost vs performance. Finding the trade-off, is known as multi-objective optimisation (MOO). Public housing stock is a major consumer of energy and is a Use Case for MOO where ***cost efficiency*** is required to optimally decide how and where to spend money on thermal efficiency upgrades, that involves making millions of decisions using cost benefit analyses to decide the mix of upgrades to sustainably deploy across thousands of housing units. This is a large scale global optimisation (LSGO), or large scale optimisation (LSO). Multi-objective evolutionary algorithms (MOEA) are one the approaches of solving these LSGO problems; and is the one that this project used.

Whilst metaheuristics-based MOEA efficacy inspires confidence in multi-variable automated decision making solutions to LSGO problems, it is often not possible to pin-point the specific factors in genetic hyperparameter configurations that are responsible for efficacy. Moreover, the effect of discrete compositions of multi-variable solutions resulting from MOEA, can serve to further cloud the picture.

Therefore, for this research project, the primary goal was to develop software in jMetalPy with MOEA for multi-variable decision making. To best align with the public housing stock *Use Case* that is rooted in buildings engineering, Deb et.al's [1] elitist non-sorting genetic algorithm (NSGA-II) was used because of its wide application in housing retrofit [2] and its ability to identify metaheuristic factors that influence MOEA efficacy vs objective functions, ***f(x)***, i.e.; cost and operational energy consumption, e.g.; genetic hyperparameter configurations, problem variables and quality of MOEA solutions.

Alongside the literature review and investigating the jMetalPy framework, initial data exploration was done using data science techniques to fix any anomalies and get the data validated. Separability was used to form a 3-tier LSGO global-problem decomposition i.e.; single house problem (SHP), group of houses problem (GHP) and full housing stock (HSP). NSGA-II was run sequentially for each granular genome from the Cambridge Housing Model (CHM) data subset. Results were captured, collated and stored, including in CSV's, at each step sub-process optimisation step. Pareto Front (PF) visualisations were prepared of the non-dominated solutions.

Traditional software engineering augmented computational intelligence (CI) to develop the metaheuristics-based MOEA. Mathematical techniques were used e.g.; ***MaxDiff*** to calculate the best point on the PF trade-off, also referred to as the pareto-optimal solution; ***ArgMax*** to find and match the last item in arrays, to its value in a corresponding array (**x** *vs* **y** *arrays for plotting graphs*); ***FindMax*** to find the reference point (RP) i.e.; maximum values for the MOEA objective functions. RP was in turn used in GHP Quality Indicator (QI) experiments to calculate efficacy of discrete MOEA genetic hyperparameter configurations, helping to address causal inference in the explainability goal. Post MOEA, mathematical techniques were used to find the ***extrema*** solutions e.g.; ***minima***, ***maxima*** and ***knee points***; and statistical techniques to find ***means***, ***distributions***, ***error rates*** and ***standard deviations***.

Results were tabulated, visualisations created, observations made, analysed and findings interpreted and explained. For example, the key findings were that it is possible to develop MOEA for LSGO in jMetalPy and; the effectiveness of the constituent intervention options in GHP solutions had more significance on MOEA efficacy, than metaheuristics, in turn suggesting that non-metaheuristics factors e.g.; speed of convergence and encoding strategy have greater significance than metaheuristics.

There were more interesting findings and suggestions for future work.

Keywords: LSGO, MOEA, NSGA-II, separability, genetic hyperparameters, explainability, hypervolume.

Attestation

I understand the nature of plagiarism, and I am aware of the University's policy on this.

I certify that this dissertation reports original work by me during my university project except for the following:

- The tools used as described in Section 5.1 are my own and were used in accordance with their licence.
- The python code used for this project was developed by me, other than the Python Frameworks and Libraries described in Section 5.1.1.
- The anonymised dataset used by this project was based on the Cambridge Housing Model (CHM) and was provided by my M.Sc. Dissertation Supervisor at the University of Stirling.

Signature John Flavian Jabbo **Date** 18 October 2021

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr Alexander (Sandy) Brownlee for his invaluable support particularly with the dissertation project. His guidance, motivation and encouragement that was offered with enthusiasm, patience and consideration for my ideas during the research and experimentation stages of the dissertation project, and suggestions of good sources to improve my knowledge during the course of this project.

I would also like to thank my personal Tutor Dr Kevin Swingler, who is also the Head of the Computing Science and Mathematics Division at the University of Stirling, for his pastoral care, throughout the course that was entirely conducted ding the period of COVID19 disruption. He ameliorated the many challenges faced by all students with online engagement manageable.

I would also like to thank all my Professors and Lecturers in the Division of Computing Science and Mathematics at the University of Stirling, for introducing me to many theoretical, applied and practical concepts that I have learned on the M.Sc. in Artificial Intelligence (AI) course. I have been able to apply a significant amount of learning from their instruction in the modules they taught, to this consequential dissertation project.

I acknowledge the contributors to Python and developers of the jMetalPy, Pandas, NumPy, Matplotlib and the others computational frameworks that I used in this dissertation project, and I am grateful for their invaluable contribution to computational data science.

I acknowledge the invaluable support from many friends outside the University, the Computing and Mathematics postgraduate student body and off campus bodies and organisations that sustained me before and during my time on the M.Sc. in AI course. They are too many to name individually.

Finally, I would like to thank all those who have directly, indirectly and in spirit, given me encouragement and inspiration in this academic endeavour and particularly in the undertaking of this consequential dissertation project.

Table of Contents

List of Figures

1 Introduction

1.1 Background

This project extended an existing evolutionary algorithm (EA) of its genetic algorithm (GA) sub-class and of the multi-objective evolutionary algorithm (MOEA) variety was previously used by Brownlee et.al [3] to solve a large scale global optimisation (LSGO) problem. LSGO is also known as large scale optimisation (LSO). EA is a super-class of metaheuristics-based artificial intelligence (AI) methods [4], and GA are a sub-class of EA.

In a real-world Use Case, a local authority in England posed a housing upgrades LSGO problem, to a multi-disciplinary team of housing improvement and LSGO researchers at Loughborough University.

The original application (Use Case) was about finding the best home insulation option or combination of home insulations options to target to particular units, in the allocation of funds for improving the overall energy efficiency of several thousand units of public housing stock. The UK Department for Business, Energy and Industrial Strategy (BEIS) [5] describes the district heating scheme (DHS) intervention option as a "*distribution system of insulated pipes that takes heat from a central source and delivers it to a number of domestic or non-domestic buildings. The heat source might be a facility that provides a dedicated supply to the heat network, such as a combined heat and power plant; or heat recovered from industry and urban infrastructure, canals and rivers, or energy from waste plants.*" DHS, therefore, is when a group of houses are connected to a shared heat source, which because of its scale, can be run more efficiently than many small-scale boilers. DHS improvements are not considered in isolation, so they greatly increase sub-problem complexity.

The data used by the existing algorithm for the LSGO problem, developed in jMetal's Java framework, did not include DHS. The algorithm used exhaustive search to solve sub-problems at the individual housing level before using a metaheuristic-based search optimisation to solve the global-problem at the stock level. Energy modelling and housing stock data was based on the Cambridge Housing Model (CHM), a free and open access UK government public housing stock dataset published by the BEIS.

In this context, the academic interest for this project is two-fold; on one hand it is to help inform real-world housing policy making, whilst on the other hand, to simultaneously develop metaheuristics-based MOEA software, that is effective in solving LSGO problems. The latter is the main focus.

1.2 Scope and Project Goals

This project's main purpose was to create software using Python to optimise the allocation of funds for improving thermal efficiency of public housing stock, extending an existing MOEA, using *separability* to sub-divide the global-problem into much larger sub-problems than those previously researched e.g., an additional group level intervention option named '*district heating scheme*' ('*dhs*') led to the creation of an intermediate group level decomposition tier named the group of houses problem (GHP), enabling an upgrade to be applied to multiple houses in a group, rather than to single houses in isolation, as previously been done by Brownlee et.al [3].

To systematically solve LSGO problems that satisfy the MOEA objectives; to (a) minimise energy consumption and (b) minimise intervention costs, this project would use a subset of free and open access UK government public housing stock anonymised datasets.

This project aimed to achieve its goals by the following means;

1. Use a systematic approach to develop metaheuristics-based MOEA [6] for multi-variable automated decision making; covering algorithmic design, simulation, experimentation and drawing utilitarian causal inference from derived results, by utilising a combination of;
    a. Traditional software engineering techniques.
    b. Computational intelligence (CI) with metaheuristics.

2. Use MOEA against CHM based dataset, to find *non-dominated solutions* that satisfy the MOEA objectives and present the results as on Pareto Front's (PF's) for each sub-problem; i.e.; single house sub-problem (SHP) and GHP.
3. Identify the *non-dominated solutions* with the best trade-off between the MOEA objectives for each SHP and GHP PF, i.e.; the ***pareto-optimal solution***, also known as the ***knee-point***.
4. Explore metaheuristics impacts i.e.; of *genetic hyperparameter configurations* on MOEA, using Deb et.al's [1] non-sorting genetic algorithm (NSGA-II), on PF points of interest i.e.; the ***extrema*** *non-dominated solutions* (***minima***, ***knee-point*** and ***maxima)***;
    a. For resultant thermal efficiency refurbishment GHP interventions.
    b. Of resultant non-functional performance e.g.; CPU time or runtime.
5. Explore MOEA execution outcomes and efficacy results, to identify the factors e.g.; *sensitivity* or *bias* in *genetic hyperparameter configurations*, that are mainly responsible for;
    a. Stochastic search optimisation efficacy.
    b. Best point on the PF trade-off, between the conflictive MOEA objectives.
    c. Variability in MOEA utilisation of computational resources.
6. Assess MOEA efficacy by extending and running jMetalPy's Experiment (*constructor super-class*), then rank the *genetic hyperparameter configurations* performance by hypervolume (HV).

1.3 Research Questions

In the context above, the purpose of this project was to address the following research questions;

1. **RQ-1** – Can LSGO problems be solved with MOEA developed in jMetalPy?
2. **RQ-2** – Can MOEA be developed in jMetalPy to provide satisfactory efficacy and reliability in its outcomes for the housing stock problem?
3. **RQ-3** – Can MOEA be developed in jMetalPy, incorporating satisfactory explainability to offer confidence in MOEA integrity in solving the housing stock problem?
4. **RQ-4** – What factors influence the efficacy and reliability of MOEA developed with jMetalPy?

1.4 Achievements

Based on the background in Section 1.1 that was outlined in Chapter 2, to provide the context for this project's original goals set out in Section 1.2, this project achieved its goals.

Initial exploratory data analysis was performed using data science techniques in Python concurrently with conceptualisation of algorithmic design and the software development approach, as described in Section 5.4. This project shows that MOEA software multi-variable automated decision making for LSGO can be developed in jMetalPy, incorporating QI's to assess MOEA and metaheuristics efficacy.

Using repeatable steps and explainability, the MOEA software was developed as seven (7) Jupyter Notebooks, with each created to perform discrete tasks, addressing scope and various specified criteria set out in Section 1.2, Section 1.3, Section 5.4 and Section 5.5, specifically;

1. Ingest the data, perform exploratory data analysis, data validation checks and data cleansing.
2. Develop and operate the MOEA software for SHP.
3. Develop and operate the MOEA software for GHP and for QI's experiments.
4. Add runtime timings for discrete tasks to explain non-functional performance.
5. Analyse *individual* and *aggregated* PF results at *genetic hyperparameter configuration* and GHP levels and create visualisations of the resultant analysis to simplify feedback to make causal inferences.

6. Analyse *individual* and *aggregated* QI's results i.e.; HV at *genetic hyperparameter configuration* and GHP levels and create visualisations of the resultant analysis to simplify feedback to make causal inferences.
7. Use *explainability* in analysis reviews, through addition of metadata to filenames, which can help drill-down by LSGO researchers and housing policy makers alike.

A systematic approach was used in software development to create repeatable iterative steps to satisfy the explainability goal, highlighting causal inference relative vs this project's outcomes, by;

1. Running all 7 Notebooks several times on two separate O/S, with; alternate dataset subsets, operating conditions, and *genetic hyperparameter configurations*, to study impacts.
2. Automation of iterative NSGA-II function calls, to create SHP and GHP PF's, as well as collation and collection of functional and non-functional results data at key stages across the MOEA lifecycle, from which on-demand analysis to obtain further resultant insights can be performed to facilitate multi-variable automated decision making.
3. Incrementing the number of variables i.e.; intervention options. '*dhs*' was added as an intervention option to each single house configuration in the initial dataset.
4. Statistical significance testing to assess MOEA efficacy, using HV scores and runtime.
5. Creating a jMetalPy based MOEA model from which future research into LSGO can be done.

Further achievements from some interesting and noteworthy findings were derived, including;

1. Enforced constraints unduly influenced the distributions of interventions applied in *non-dominated solutions*, particularly at the GHP level. For example, due to a combination of factors, there were no '*zero solutions*' on the PF's, because every house had at least one intervention option applied, as described in Section 7.2.
2. Composition of constituent interventions in GHP solutions had a much more significant impact on MOEA efficacy than metaheuristics, as described in Section 7.2.

My knowledge and skills in optimisation, LSGO and MOEA have significantly improved, as has my applied mathematical and statistical techniques in programming. Using QI to assess algorithm efficacy has enabled me to contribute to LSGO decomposition research, and emerging area of research with pertinence to housing stock optimisation.

There were setbacks (*see Section 8.3 and Section 8.4*). However, identification of these issues were in themselves, important findings, as their resolution in future shall further this applied research work.

On reflection, overall, this project's achievements have far exceeded the goals set out at the outset; as it includes many suggestions for future work beyond the project brief.

1.5 Code Repository

The project artefacts have been archived in the university's digital project repository.

Details of these artefacts are listed in Appendix 1.

1.6 Overview of Dissertation

This project's work has been organized into eight chapters as follows:

Chapter 1: Introduction sets out the problem statement, project motivations, and explains why this was worthwhile work to do. It also describes this project's scope and goals, and then summarises this project's achievements.

Chapter 2: Research Background and Situational Context describes this project's motivation from the perspectives of both LSGO with MOEA as well as housing improvement, and then introduces relevant State-of-the-Art in this regard, that are then critically examined in the next chapter.

Chapter 3: State-of-the-Art discuses domain research in the development and use of MOEA for LSGO and elucidates upon recent developments in this area of research, and in its application to LSGO for housing improvements involving large scale housing stock.

Chapter 4: LSGO Problem Decomposition describes the theoretical concepts behind the approach to solving LSGO problems. It includes the comparative conceptualisation of separability techniques used in comparable work that were built upon and used by this project vs prior orthodox exhaustive search optimisation techniques and addresses explainability of the theoretical concepts.

Chapter 5: Methodology describes the tools, methodologies used and this project's approach to research and investigation for both LSGO and MOEA. It also covers exploratory data analysis and data preparation. It highlights how the methodologies were used in conjunction during the iterative investigative work of this project and making rationale for choices made with both methodology and subsequent software development.

Chapter 6: Software Development using MOEA in jMetalPy describes how the software was designed and built using computational intelligence (CI) augmented by traditional software engineering techniques. It further describes how statistical techniques were used for the analysis of results.

Chapter 7: Results and Discussion describes the results obtained from various investigations and explains the observations made from the results and insights derived from analysis of the results vis-a-vis this project's goals set out in Section 1.2, and discusses the significance of notable observations and insights derived on MOEA use for LSGO automated multi-variable decision making.

Chapter 8: Conclusions summarises the overall findings and recommendations from this project, and then describes what was achieved vis-à-vis not achieved, and then makes suggestions for Future Work in terms of academic scientific research and further investigations. It further describes the challenges faced during this project.

2 Research Background and Situational Context

This chapter describes the background, perspective and context from which this project evolved, starting with the branch of mathematics that is optimisation and addresses the circumstances that have given inspiration for this research project.

2.1 Multi-Objective Optimisation (MOO)

Resources are finite in the real world. Optimisation is the multi-disciplinary practise of finding the most effective way to utilise finite resources [7], using mathematical procedures to maximise or minimise the objective functions [8]. Many scholars have used optimisation to sustainably utilise resources, e.g.; Michailos et.al [9], for energy utilisation in ethanol production.

Eiben et.al [10], [11] and Gad [12] describe the four main categories of optimisation as;

1. Constrained Optimisation.
2. Multi-Modal Optimisation.
3. Multi-Objective Optimisation (MOO).
4. Combinatorial Optimisation.

MOO is a sub-field of optimisation that identifies viable solutions that best satisfy a multiple number of conflictive objective functions ***f(x)*** for a problem amongst a pool of possible solutions [8]. Chang [13] described alternative ways that MOO is referred to including; (a) multi-objective programming (MOP), (b) vector optimisation, (c) multi-criteria optimisation (MCO), (d) multi-attribute optimisation (MAO), and (e) pareto optimisation.

Manson et.al [14] describe ***variables*** in MOO as discrete features (*columns*) in data that are used to optimise MOEA objectives. In the context of separability for MOEA, Ma et.al [15] describe ***decision-variables*** as multitudes of several low-dimensional sub-components (*akin to elements in solutions*) and further describe ***variables*** as independent of other ***variables***, contradicting Yu et.al's [16] earlier research that describes ***variables*** as inter-dependent for difficult optimisation problems e.g.; LSGO and by extension, MOEA, suggesting that separability changes how LSGO treats ***variables*** and ***decision-variables***.

As shown in Figure 2, dominance in MOO solutions is when no other solution is better than the specified solution for all MOO objectives. When joined together on the PF, *non-dominated solutions* [17] form a line that referred to as the pareto curve (PC). It is from *non-dominated solutions* that choices are made for solutions to proceed with, depending on prevailing priorities. The *non-dominated solution* on the PC that offers the best trade-off between the conflictive MOO objectives, is the pareto-optimal solution [17] and is usually referred to as the knee-point, as shown in Figure 1 and Figure 2 below.

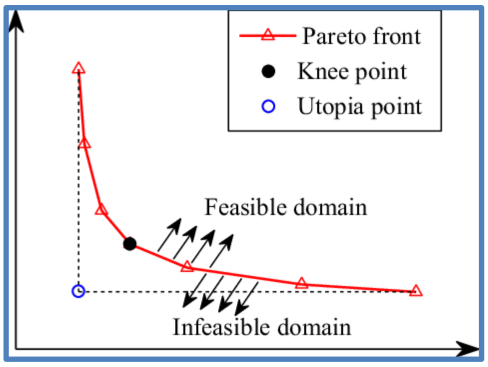


Figure 1. Conceptual PF with Knee Point [18].

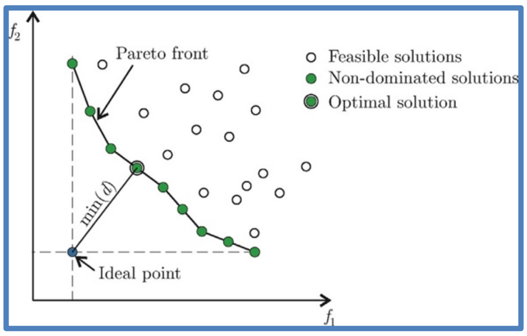


Figure 2. Conceptual PF showing Optimal Solution [17].

This project's focus is about MOO, as applied to LSGO. In particular, to MOEA, which is a way to solve MOO using EA, with problem decomposition. Due to its wide application in housing retrofits [2] and ability to identify metaheuristic factors that influence MOEA efficacy e.g.; genetic hyperparameter configurations, problem variables and quality of solutions, NSGA-II was used as the MOEA of choice.

The MOEA framework in Python that was used was jMetalPy v1.5.5.

2.2 Project Background

Inspiration for this research project is rooted in prior building physics research by a multi-disciplinary team from Loughborough University's School of Architecture and Civil Engineering, and the Division of Computing Science and Mathematics at the University of Stirling, who developed a novel approach to LSGO problem formulation [19] and MOO models/simulations with jMetal's Java framework [20]. Using English Housing Survey (EHS) data; the researchers solved LSGO to find cost-effective combinations of thermal efficiency refurbishment options for local government housing retrofit programmes.

This project used CI to develop MOEA software to solve LSGO for a public housing stock Use Case, that incorporated metaheuristics-based optimisation [6]. This built upon prior research [19], [20] and complimentary building stock optimisation research by Wright et.al [21] and Brownlee et.al [3] who proposed a novel approach to LSGO that uses *combinatorial separability* to MOO problems with several vast search spaces such as the ones dealt with by this project in the magnitude of *5.00e+31* for just one of thousands of NSGA-II runs. The Use Case was public housing stock optimisation in England.

This project exploited prior LSGO knowledge to introduce *combinatorial separability* to solve LSGO, with a 3-tier approach to problem decomposition, i.e.; SHP, GHP and full housing stock (HSP). Both SHP and GHP are sub-problems, with HSP as the global-problem.

Prior knowledge of the problem is exploited to extend the use of separability [3] and CI to develop self-adaptive GA [22] in jMetalPy, that learn and adjust by themselves in a manner akin to unsupervised learning in machine learning (ML), to reconfigure control parameters in the MOEA genetic hyperparameters (*i.e.; population, mutation, crossover, selection and termination*). As a result, the MOEA introduces process and computational efficiency by reducing the need to duplicate tasks by solving simpler problem before subsequently tackling the more complex problems. As with Brownlee et.al. [3], this project used separability to formulate the problem by diving it into the aforementioned SHP and GHP, thus dividing the search spaces into still vast but smaller and more manageable pieces.

Whilst there has been extensive research into MOEA to solve LSGO, the use of *combinatorial separability* to LSGO of large housing stock has not been reported outside the aforementioned Loughborough based research group, making this a relatively new area research into LSGO.

Zille et.al [23] used dimensionality reduction techniques to the search space to simplify computation and reduce computational effort for numerical benchmark functions, whilst Cabrera's [24] large overarching study concluded that memetic algorithms dealt best with LSGO after reviewing research papers to identify trends in LSGO where vast search space sizes and a variety of decomposition techniques in co-evolutionary algorithms with several decision-variables. Notably, however, none of these efforts in use separability to simplify computation by creating discrete sub-problems as Brownlee et.al [3] and this project have done.

Unlike prior research, this project incorporates explainability in the code and in the results produced, to provide the foundation to deduce insights to addresses the research questions posed, e.g.; using a dictionary of lists of multiple genetic operator values used to automate the selection of genetic hyperparameter configurations, enabling the identification of the best MOEA convergence [25] and in turn the GHP solutions that have the highest HV scores.

This project's artefacts are adding to the growing LSGO body of research, particularly with regards to (a) LSGO problem formulation or decomposition through separability, (b) determining MOEA efficacy through QI's and solution composition, (c) and the use of jMetalPy as a platform for research that is dependent on metaheuristics-based MOEA.

2.3 Housing Improvement Motivations

Housing improvement motivations for this project were;

1. To simplify the identification of thermal efficiency refurbishment options for thousands of units of aging and thermally inefficient public housing stock, including providing ability to maximise prioritisation by identifying trade-off between viable solutions.
2. To use LSGO to support public housing policy decision making and optimise the utilisation public funds for refit, refurbishments or new build designs.
3. To facilitate effective targeting of public funds to prevailing priorities e.g.; design of future buildings to reduce non-renewable energy consumption or to achieving climate and low carbon targets e.g.; low emissions, net zero, or investing in renewable energy.

2.4 Genetic Improvement Motivations

The genetic improvement motivations for this project were;

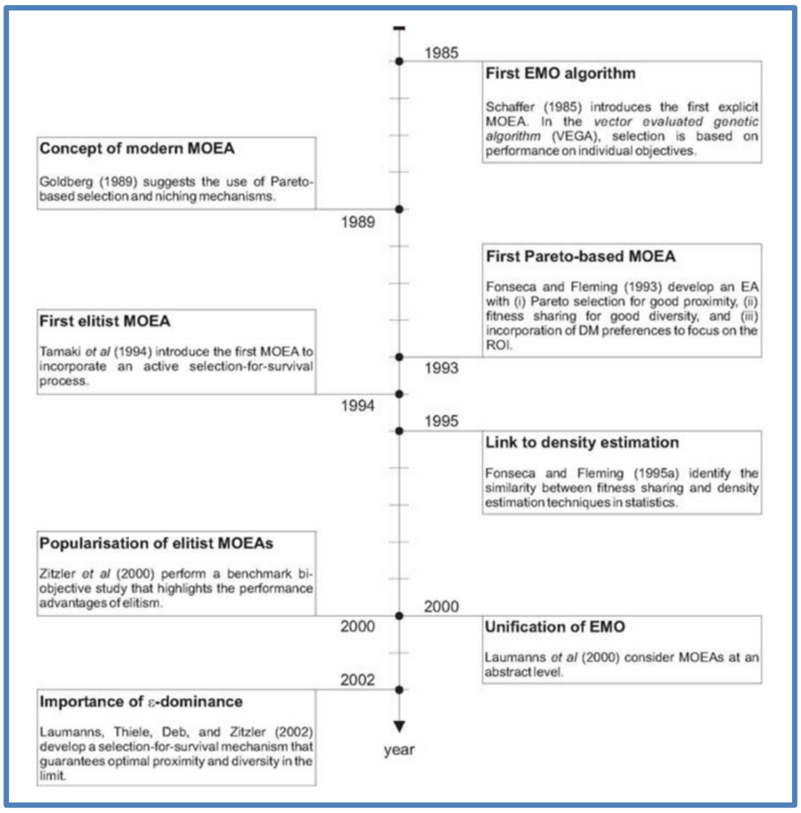
1. To incorporate problem decomposition and separability in MOEA algorithmic design.
2. To use CI including metaheuristics to optimise search space and improve MOEA efficacy through computational resource optimisation.
3. To study functional and non-functional factors that influence MOEA sensitivity and bias e.g.;
    a. Stochastic search optimisation efficacy.
    b. Trade-off between multiple competing MOEA objectives.
    c. Variability in MOEA utilisation of computational resources.
4. To use data science techniques for exploratory data analysis in order to identify and address anomalies in the data and update the dataset from a building physics model to MOEA ready datasets for the separate sub-problems.
5. To use statistical causal inference to explain the key factors that determine MOEA outcomes. For example, to understand how these factors affect the ***extrema*** solutions on the PF e.g.;
    a. ***Minima***: the 'lowest energy used' but highest cost.
    b. ***Knee point***: the best trade-off between 'energy used' and 'cost'.
    c. ***Maxima***: the 'highest energy used' but lowest cost.
6. To use explainability in the inferences that determine of this project's findings, conclusions and subsequent recommendations.

3 State-of-The-Art

This chapter reviews state-of-the-art research into MOEA for LSGO housing stock optimisation.

3.1 Chronological Background to NSGA-II, MOEA

Evolutionary Multi-Criteria Optimisation (EMO) first proposed in 1985 by Schaffer, and then MOEA was proposed in 1989 by Goldberg, then various revisions followed until 2002, when Deb et.al [1] proposed the elitist NSGA-II, as illustrated in Figure 3 below.



1985
First EMO algorithm
Schaffer (1985) introduces the first explicit MOEA. In the *vector evaluated genetic algorithm* (VEGA), selection is based on performance on individual objectives.
Concept of modern MOEA
Goldberg (1989) suggests the use of Pareto-based selection and niching mechanisms.
1989
First Pareto-based MOEA
Fonseca and Fleming (1993) develop an EA with (i) Pareto selection for good proximity, (ii) fitness sharing for good diversity, and (iii) incorporation of DM preferences to focus on the ROI.
First elitist MOEA
Tamaki *et al* (1994) introduce the first MOEA to incorporate an active selection-for-survival process.
1993
1994
1995
Link to density estimation
Fonseca and Fleming (1995a) identify the similarity between fitness sharing and density estimation techniques in statistics.
Popularisation of elitist MOEAs
Zitzler *et al* (2000) perform a benchmark bi-objective study that highlights the performance advantages of elitism.
2000
2000
Unification of EMO
Laumanns *et al* (2000) consider MOEAs at an abstract level.
2002
Importance of ε-dominance
Laumanns, Thiele, Deb, and Zitzler (2002) develop a selection-for-survival mechanism that guarantees optimal proximity and diversity in the limit.
year

Figure 3. Key developments in MOEA history [26]

The background to NSGA-II illustrated in Figure 3 above, is advanced by this project and the research work that inspired it that is outlined in Section 2.2 and Section 2.3 above, through the incorporation of separability in the way that NSGA-II work as part of LSGO problem decomposition.

3.2 Computational Intelligence (CI)

Rather than solely use conventional linearity based search optimisation, CI draws from AI to additionally use randomness in processes and often on nature inspired approaches to learning, including metaheuristics, artificial neural networks and fuzzy systems, in combination with traditional rules-based computation. This trend has recently gained pace in the desire to solve complex problems that require either significant computational effort or more efficient stochastic search optimisation to propagate through vast search spaces, as evidenced by Venture Capital interests funding the adoption of an new approach by Google's DeepMind, that '*reconciles Deep Learning and classical computer science algorithms with Neural Algorithmic Reasoning*' i.e.; combining traditional software engineering with ML and deep learning (DL), with the ultimate aim to help increase the uptake of Neural Networks in many more commercial Use Cases, as reported by Anadiotis [27]. As in the DeepMind example [27], this project combines traditional rules-based computation with nature inspired metaheuristics to solve and LSGO problem as espoused by the illustrations in Figure 4 and Figure 5 below.

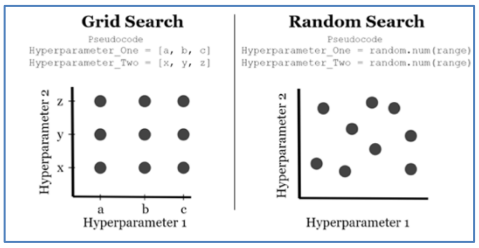

Grid Search
Pseudocode
Hyperparameter_One = [a, b, c]
Hyperparameter_Two = [x, y, z]
Hyperparameter 2
Hyperparameter 1
Random Search
Pseudocode
Hyperparameter_One = random.num(range)
Hyperparameter_Two = random.num(range)
Hyperparameter 2
Hyperparameter 1

Figure 4. Computation with Search – Grid vs Random

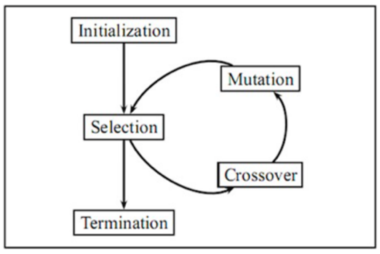


Figure 5. Computation with EA/GA [6]

The family of algorithms that use metaheuristics are known as EA as categorised by Azad et.al [28] from work done by earlier researchers, in Figure 6 below.

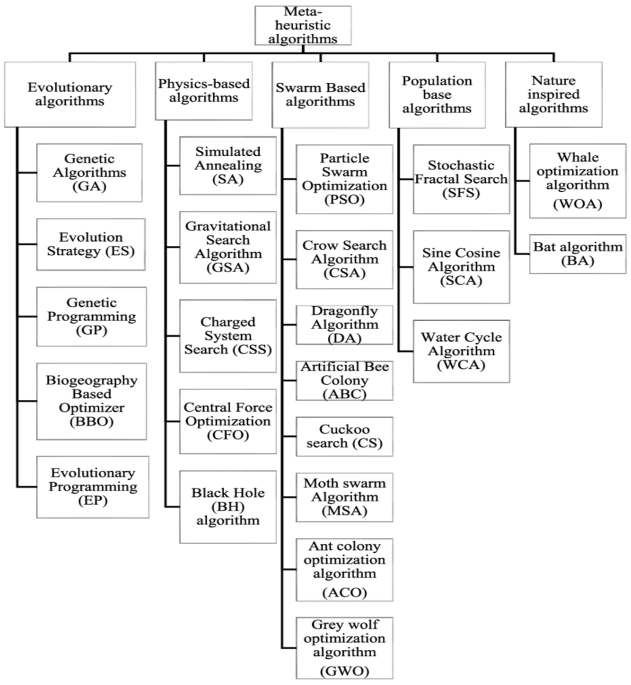

Meta-heuristic algorithms
Evolutionary algorithms
Physics-based algorithms
Swarm Based algorithms
Population base algorithms
Nature inspired algorithms
Genetic Algorithms (GA)
Evolution Strategy (ES)
Genetic Programming (GP)
Biogeography Based Optimizer (BBO)
Evolutionary Programming (EP)
Simulated Annealing (SA)
Gravitational Search Algorithm (GSA)
Charged System Search (CSS)
Central Force Optimization (CFO)
Black Hole (BH) algorithm
Particle Swarm Optimization (PSO)
Crow Search Algorithm (CSA)
Dragonfly Algorithm (DA)
Artificial Bee Colony (ABC)
Cuckoo search (CS)
Moth swarm Algorithm (MSA)
Ant colony optimization algorithm (ACO)
Grey wolf optimization algorithm (GWO)
Stochastic Fractal Search (SFS)
Sine Cosine Algorithm (SCA)
Water Cycle Algorithm (WCA)
Whale optimization algorithm (WOA)
Bat algorithm (BA)

Figure 6. Categorisation of Metaheuristic Algorithms [28]

This project's research focus is using metaheuristics-based MOEA to develop software to solve LSGO, with public housing improvements as the Use Case, as outlined by Azad et.al [28] in Figure 7 below.

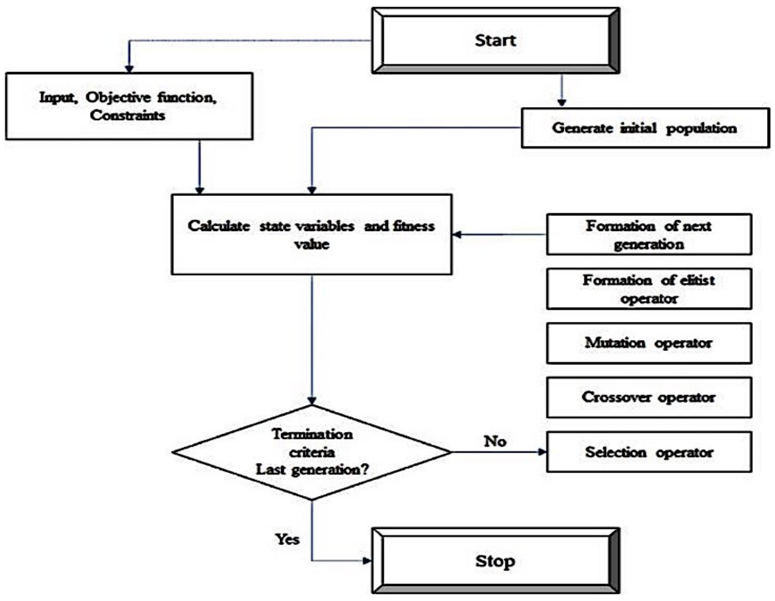

Start
Input, Objective function, Constraints
Generate initial population
Calculate state variables and fitness value
Formation of next generation
Formation of elitist operator
Mutation operator
Crossover operator
Selection operator
Termination criteria Last generation?
No
Yes
Stop

Figure 7. Flowchart of a Metaheuristics-based MOEA [28]

The NSGA-II variant of MOEA used by this project is consistent with both the illustration by Azad et.al [28] in Figure 7 above, with the recent CI approach used by DeepMind [27].

An advantage of MOEA is that there are reduced mathematical coding requirements than tradition software engineering would otherwise require [29], which is in turn further advantageous as it simplifies the coding for mathematically complex problems that can be solved efficiently with less computational effort to achieve the same level of efficacy as maximum linear computational effort would, e.g.; runtime vs search space size. Another advantage/benefit is that unlike traditional software engineering, MOEA can achieve parallelism in its implementation without significant programming effort.

A notable disadvantage is MOEA efficacy is typically dependent on local knowledge of the Use Case.

3.3 Literature Review

Researchers have studied many EA/GA to solve LSGO problems, including simulated annealing, NSGA-II, NSGA-III, SMPSO, OMOPSO, MOEA/D, MOEA/D-DRA, εMOEA, GDE3, SPEA2, HYPE, IBEA which are implemented in jMetalPy.

This project only considered MOEA/D and NSGA-II, eventually choosing to only use NSGA-II.

Problem decomposition has become a central feature of LSGO computational simulation. In their LSGO decision variable research, Ma et.al [15] decompose complicated multi-objective problems (MOP) into simpler sub-MOP's as long as each variable is independent and can be optimised separately, in a manner that has similarities to Brownlee et.al's [3] approach to separability for LSGO. Ma et.al [15] further describe the objective function f(x) as a separable function, in such circumstances, or as a non-separable function where the independence of variables is not possible.

Brownlee et.al's [3] novel encoding approach uses two stages;

- Stage-2: LSGO global-problem (*entire housing stock*) resulting in 1 LSGO PF.
- Stage-1: SHP sub-problem (*single houses*) resulting in 935 SHP PF's.

The viable SHP solutions (on PF's) to the sub-problems at Stage-1 are used to find a solution more efficiently at Stage-2. Without Stage-1, Stage-2 is intractable.

For this project on the other hand, *an intermediate stage (GHP) was necessary because of the addition of the 'dhs' intervention option, enforced a constraint that can only be implemented at GHP level*. Therefore, a third stage (Stage-3) would be required to optimise the entire housing stock as illustrated by the steps below;

- Stage-3: LSGO global-problem, HSP (*entire housing stock*) *not done because it was intractable*
- Stage-2: GHP sub-problem (*groups of houses*) resulting in 31 GHP PF's.
- Stage-1: SHP sub-problem (*single houses*) resulting in 935 SHP PF's.

SHP solutions (PF's) to the Stage-1 sub-problems, are used to find GHP solutions more efficiently at Stage-2, sequentially for each of the 31 sub-problems at Stage-2. Without Stage-1, Stage-2 is intractable, and without Stage-2, Stage-3 is intractable.

Therefore, to solve the LSGO global-problem using Stage-3 (*ascribed by this project to Future Work*) would involve using GHP solutions (GHP PF's) to find the global-solution more efficiently at Stage-3, and in the process solve the LSGO even more efficiently than Brownlee et.al [3], who didn't benefit from the intermediate stage.

Stage-3 implementation shall involve creating a new jMetalPy constructor super-class (*HSP*) with supporting abstract sub-classes in jMetalPy, decomposed with a value encoding (*IntegerProblem*) strategy. The 31 GHP PF's data (*from PF Index column in the GHP Results*) would be used as inputs to the NSGA-II *problem* parameter, to sequentially run the 31 NSGA-II iterations of the *HSP* Stage-3 problem to create a single resultant PF for the LSGO global-problem (*HSP*).

Brownlee et.al [30] argue the prevailing deficiency in explainability for metaheuristics-based EA and offer an approach to explain MOEA efficacy by identifying the genetic hyperparameter configuration performance. This project offers a similar but different approach that additionally considers the constituents makeup of the *non-dominated solutions*. Indeed, this project found that the makeup of solutions had a greater impact on MOEA efficacy than *genetic hyperparameter configurations*.

With the exception of Brownlee et.al's [3] research, the typical scale of building optimisation problems has tended to be in the magnitude of hundreds of decision-variables and not in the millions of decision-variables that this project has dealt with, as shown in Figure 34 and in Figure 35 in Sub-section 7.1.6. In their extensive building energy optimisation research, Waibel et.al [31] indeed found fewer examples of combinatorial optimisation problems as Brownlee et.al [3] and this project dealt with, further pointing to significantly smaller magnitude of decision-variables in most studies. Waibel et.al [31] likewise found the same to be true in buildings form optimisation research.

Whilst there aren't any other works in building energy optimisation that go to the scale of Brownlee et.al's [3] novel approach that uses separability, there's nonetheless a mathematically efficient approach by Westermann et.al [32] that uses statistical modelling as an alternative to computational simulations to solve LSGO housing problems. However, this is not a like-for-like comparison as Westermann et.al [32] whose approach does not use computational effort in the same way.

On another hand, in their investigation into the performance of black-box optimisers, Waibel et.al [31] identify the need to tune the parameters for black-box optimisers to elicit faster convergence which is a comparable approach to the genetic hyperparameter tuning of 64 configurations done by this project. Black box optimisers is a general term that means metaheuristics-driven optimisers.

This project used NSGA-II in jMetalPy whereas Brownlee et.al's [3] much larger study used jMetal's Java framework, and various encodings for multiple MOEA including MOEA/D, SPO and NSGA-II.

4 LSGO Problem Decomposition

In this chapter, the functional considerations in MOEA algorithm complexity are addressed.

As introduced in Section 1.1, Section 2.1 and Section 2.3, LSGO is the branch of optimisation that deals with computationally complex vast problem search spaces from which to make a vast number of decisions, in the thousands or even millions. LSGO is an appropriate solution for the public housing thermal optimisation Use Case that was introduced in Section 1.1, because it is capable of automated multi-variable decision making as is required by the Use Case.

According to Zhang et.al [29], decomposition was largely omitted from MOEA until the advent of MOEA based on decomposition (MOEA/D) that he first proposed in 2007. As observed by Li [33] whose analysis infers LSGO as a motivation for the adoption of scalar optimisation techniques to address sub-problems rather than try to solve often intractable LSGO problems.

The advent of MOEA/D parallels improvements in *explainability* which is crucial to assessments of (a) integrity in MOEA decomposition and construction; and (b) integrity in MOEA outcomes, i.e.; the *non-dominated solutions*. But why is *explainability* typically not done to the satisfaction of LSGO end-users whose understanding of MOEA is often limited by the inherent black-box characteristics [30] in metaheuristics? Li [33] argues that it could be because decomposition was not well studied MOO until the advent of MOEA based on decomposition (MOEA/D) in 2007.

This project seeks to address *explainability* through *LSGO problem decomposition*.

4.1 Problem Outline

As described in Section 5.2, the initial dataset dimensions for this project's LSGO problem comprised 27,150 rows and 37 columns, for 935 single houses in 30 separate groups of houses. Of the 37 columns, the 10 intervention options and their 10 corresponding cost columns were target variables for the MOEA objectives, for multi-variable decision making for any format of LSGO problem decomposition. Each single house was represented by a variable number of rows, each representing a unique configuration i.e.; different combination of thermal intervention options applied to that house.

After data cleansing the data dimensions of the modified dataset increased to 54,300 rows and 38 columns, for 935 single houses in 30 separate groups of houses. Of the 38 columns in the modified dataset, the 11 intervention options and their 11 corresponding cost columns were target variables for the MOEA objectives, for multi-variable decision making for any format of LSGO problem decomposition. As with the original dataset, each house in the modified dataset was represented by a variable number of rows each representing a unique configuration for that house.

This dataset creates a computational challenge due the vast search spaces created and the resultant vast number of decision-variables, in the thousands, as described in Sub-section 4.1.1 and Sub-section 4.1.2 below.

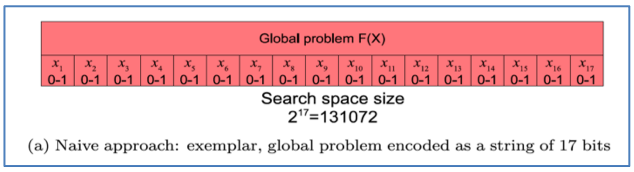
4.1.1 Naïve Approach to Search Space Optimisation

Given that the typical search space for LSGO problems that Kulkarni et.al [34] describe as '*often algorithmically solvable but computationally intractable*', this project used the novel approach to search space optimisation proposed by Brownlee et.al [3].

Following the orthodox exhaustive search approach, this project's LSGO global-problem selects an optimal combination of configurations for each house in a group of houses. A solution to the group problem is a combination of chosen configurations for each house in a particular group of houses. In this '*naïve approach*' [3], all possible LSGO solutions are considered, i.e.; all possible combinations of configurations for each house, thereby, making the search space the number of possible solutions, which is also the number of possible solution combinations. This is calculated by multiplying together the number of possible configurations for each house in the group i.e.; their product.

For this project, using the naïve approach, the LSGO problem search space size using the exemplar formula in Figure 8 below was; *9.99e+53 = 9.99 * 10^53*, compared to Brownlee et.al's novel approach where the search space size for the LSGO problem was the much smaller; *5.43e+04 = 5.43 * 10^4*.

The naïve approach yields an extremely large search space size that renders GHP an intractable problem that cannot be solved within a reasonable (a) amount of time and (a) cost, due to the required computational effort, compared to Brownlee et.al's [3] novel approach which makes it solvable.


Global problem F(X)
x_1 0-1 x_2 0-1 x_3 0-1 x_4 0-1 x_5 0-1 x_6 0-1 x_7 0-1 x_8 0-1 x_9 0-1 x_{10} 0-1 x_{11} 0-1 x_{12} 0-1 x_{13} 0-1 x_{14} 0-1 x_{15} 0-1 x_{16} 0-1 x_{17} 0-1
Search space size
2^{17}=131072

(a) Naive approach: exemplar, global problem encoded as a string of 17 bits

Figure 8. Orthodox exhaustive LSGO Search Space calculation using the '*naïve approach*' [3]

In Figure 8 above, bits represent the intervention options applied in a particular solution.

4.1.2 Novel Approach to Search Space Optimisation

It was necessary to use an alternate approach to search space optimisation, given the intractability of the naïve approach described in Sub-section 4.1.1. To this end, Brownlee et.al's [3] metaheuristics friendly novel approach to breakdown the global LSGO problem into two sub-problems was adopted, as it directly addressed the project goals set out in Section 1.2 and Section 1.3.

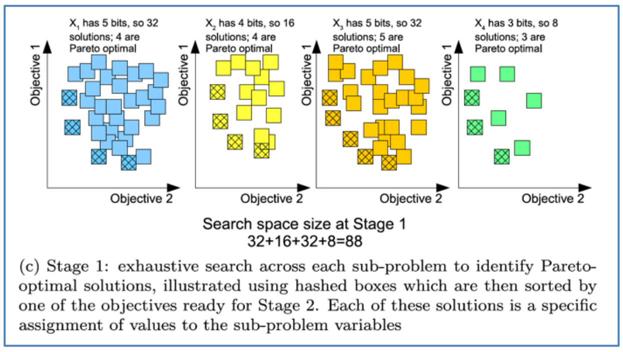
As used by Brownlee et.al [3], the LSGO problem was broken down in two-stages;

1. Stage-1 – SHP sub-problem.
2. Stage-2 – GHP sub-problem.

In a clear distinction from the naïve approach, the search space size using Brownlee et.al's novel approach [3], is the sum of each of the search space sizes of the individual sub-problems, rather than their product, as applied by the exhaustive naïve approach.

4.1.2.1 Stage-1 – SHP Decomposition

In Stage-1, SHP sub-problem optimises individual sub-problems sequentially in a manner reminiscent of the Sequential Pareto Optimisation (SPO), i.e.; by solving each single SHP sub-problem separately. Each SHP sub-problem is solved with an SHP PF for that particular house. The search space size of the SHP sub-problem is the sum of each of the individual search spaces of the separate SHP sub-problems. In other words, the SHP search space size is the sum of the number of *separate non-dominated house configurations/solutions* that appear on the SHP PF, and specifically excludes *dominated configurations/solutions*, as illustrated by Brownlee et.al [3] in Figure 9 below.



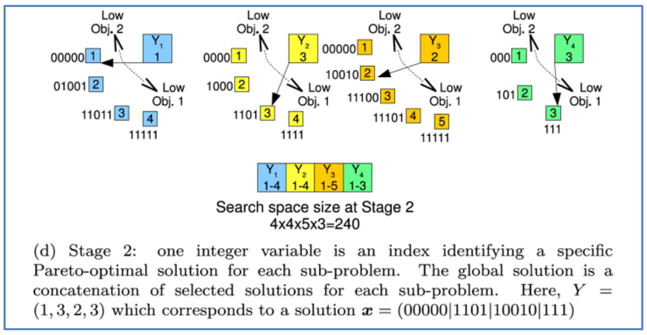
(c) Stage 1: exhaustive search across each sub-problem to identify Pareto-optimal solutions, illustrated using hashed boxes which are then sorted by one of the objectives ready for Stage 2. Each of these solutions is a specific assignment of values to the sub-problem variables

Figure 9. Stage-1 – SHP Search Space Size using '*Novel Approach*' by Brownlee et.al [3]

In Figure 9 above, bits represent the intervention options applied in a particular SHP solution.

4.1.2.2 Stage-2 – GHP Decomposition

In Stage-2, GHP sub-problem builds upon SHP sub-problem results to optimise individual sub-problems sequentially, i.e.; by solving each single GHP sub-problem separately. Each GHP sub-problem is solved with a GHP PF for that particular group of houses. The search space size of the LSGO GHP sub-problem is the sum of each of the individual search spaces of the separate GHP sub-problems. In other words, the GHP search space size is the sum of the number of *separate non-dominated group of houses configurations* that appear on the GHP PF, specifically excluding *dominated configurations/solutions*, as illustrated by Brownlee et.al [3] in Figure 10 below.



(d) Stage 2: one integer variable is an index identifying a specific Pareto-optimal solution for each sub-problem. The global solution is a concatenation of selected solutions for each sub-problem. Here, $Y = (1, 3, 2, 3)$ which corresponds to a solution $\boldsymbol{x} = (00000|1101|10010|111)$

Figure 10. Stage-2 – GHP Search Space Size using '*Novel Approach*' by Brownlee et.al [3]

In Figure 10 above, bits represent the intervention options applied in a particular GHP solution.

4.2 Applying Separability to LSGO

It is widely acknowledged that monolithic optimisation algorithms are deficient for LSGO, and Li [35] argued that *divide and conquer* algorithmic strategies that decompose problems into smaller more manageable pieces as the co-operative co-evolutionary (CC) first proposed by Potter and De Jong [36], or more recently implemented by Brownlee et.al [3] and also implemented by this project, are more suitable for LSGO. Li [35] further described decomposition as; '*where a problem is decomposed into several subcomponents of smaller sizes, and then each subcomponent is "cooperatively coevolved" with other subcomponents*' i.e.; multi-stage processing that solve different problems of a much larger problems in a co-operative manner.

Simply, separability is implemented by breaking down LSGO into smaller sub-problems and solving them separately to the overall benefit of solving the overarching LSGO. This perhaps inadvertently, additionally introduces simplicity and better maintainability of software.

This project setup SHP as a *Binary Encoding Problem* because the non-dominated solutions have a set of elements that can either be applied or not, hence are *Truth* values aligned to *Binary Encoding*, whilst GHP was setup as a *Value (Integer) Encoding Problem* because the non-dominated solutions are a set of embedded sets of SHP solutions and could therefore not be setup as a *Binary Encoding Problem*.

For both encoding strategies, the non-dominated solution's *viability quality* score is a unique integer starting from 1 for the best solution; and changing in increments of 1 for the next best solution and so on. The PF solutions scores are stored in the *Group PF Index* column against the corresponding *solution configuration* row; and are ranked according to their proximity to a MOEA objective function, i.e.; either to energy consumption or to the cost of thermal efficiency interventions.

4.3 LSGO Problem Configuration

The LSGO problem was configured for SHP and GHP in NSGA-II in a manner consistent with the metaheuristics principles that were described in Section 5.3 and Section 5.4.

This project chose to use the jMetalPy framework [37] for this research. To determine an appropriate high-level outline for this LSGO problem decomposition, this project used a combination of; (a) local knowledge, (b) a data model produced from prior research by Brownlee et.al [3], (c) ideas from the

Benitez-Hidalgo et.al’s introductory paper on jMetalPy [38] and (d) some further guidance from the jMetalPy framework documentation on GitHub [37] and online [39] .

The separability approach similar to Brownlee et.al [3] was used to divide the LSGO problem into two sub-problems, each using a different encoding strategy, as follows;

- SHP, using the Binary Encoding strategy.
- GHP, using the Values (*Integer*) Encoding strategy.

For each of the two sub-problems, the configuration then took the following repeatable steps using the JMetalPy source code and Application Programming Interface (API) as integrated templates for MOEA software being developed in jMetalPy;

1. Create a new constructor super-class i.e.; *BinaryProblem* for SHP or *IntegerProblem* for GHP.
    a. Modify/Add abstract sub-classes, as required.
    b. Modify/Add abstract functions, required.
2. Test the new SHP and GHP *constructor super-class* with a limited dataset.
3. Add code to iteratively;
    a. Calculate runtime for each discrete NSGA-II function call.
    b. Identify the beginning and start positions of new houses in the data.
    c. Pick dictionary values to use to evaluate new populations.
    d. Create a dictionary of genetic hyperparameter values to be used by NSGA-II.
    e. Create new data structures to hold the results data from NSGA-II function calls.
    f. Create new data structures to hold QI jobs for subsequent QI experiments.
    g. Write PF results, and other housing stock metadata to CSV.
    h. Visualise PF results and save visual results to PNG.

The additional code for data exploration and for post optimisation results analysis are described and discussed in Section 6.2 and Section 6.3.

4.3.1 NSGA-II Sub-Problem Configuration

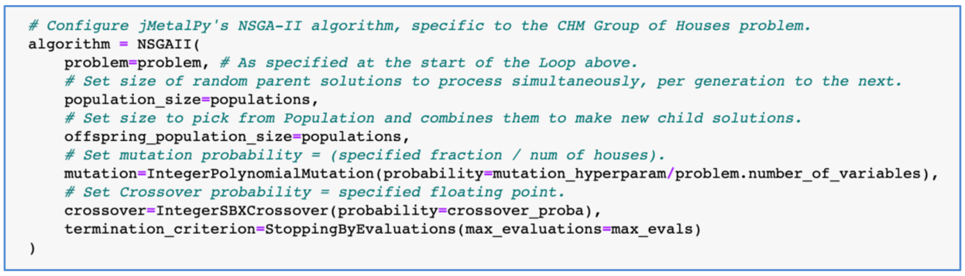
In a manner consistent with Figure 5 and Figure 7, and as shown in Figure 11 below, NSGA-II algorithm function calls were setup for both SHP and GHP using the following steps;

1. In the sub-problems super-class initialisation abstract class *__init__*, set up the following;
    - Formula for sub-problem decomposition.
    - Formula to create initial solution, with random non-zero values.
    - Number of objective functions.
    - Directions for the objective functions e.g.; maximise or minimise, as required.
    - Labels for the objective functions e.g.; Energy Used or Costs, as required.
    - Formula for number of units in the sub-problem decomposition.
    - Formula for number of variables in the sub-problem decomposition.
    - Number of constraints in the sub-problem decomposition.
    - Formula for lower bound in the sub-problem decomposition.
    - Formula for upper bound in the sub-problem decomposition.
2. In the sub-problems super-class evaluation abstract class *evaluate*, set up the following;

- Formula for the solutions to include in the sub-problems decomposition.
- Formula to enforce constraints in the sub-problems decomposition. *Bias can be enforced via constraints e.g.; lowering renewable energy costs to prioritise renewable thermal efficiency intervention options*.

3. Create additional bespoke abstract classes in the sub-problems super-class as required.
4. In NSGA-II function call, set the *population size* and *offspring population size* as the same variable *population*. They must match, hence using a single variable for both. A generation is double the population value i.e.; if 100, then 1 generation is 200.
5. In NSGA-II function call, set the mutation operator as; *BitFlipMutation* for SHP, and *IntegerPolynomialMutation* for GHP, followed by the mutation probability value, using the variable *mutation_hyperparam*.
6. In NSGA-II function call, set the crossover operator as; *SPXCrossover* for SHP and *IntegerSBXCrossover* for GHP, followed by the crossover probability value, using the variable *crossover_proba*.
7. In NSGA-II function call, set the termination criteria as; *StoppingByEvaluations*, and the set the termination criteria value the *constant* variable *max_evals*.
8. The unspecified *selection* genetic operator is automatically set to the default value *BinaryTournamentSelection(comparator=RankingAndCrowdingDistanceComparator()*.

For illustrative purposes, Figure 11 below shows setup of the NSGA-II function call for GHP.



```
# Configure jMetalPy's NSGA-II algorithm, specific to the CHM Group of Houses problem.
algorithm = NSGAII(
    problem=problem, # As specified at the start of the Loop above.
    # Set size of random parent solutions to process simultaneously, per generation to the next.
    population_size=populations,
    # Set size to pick from Population and combines them to make new child solutions.
    offspring_population_size=populations,
    # Set mutation probability = (specified fraction / num of houses).
    mutation=IntegerPolynomialMutation(probability=mutation_hyperparam/problem.number_of_variables),
    # Set Crossover probability = specified floating point.
    crossover=IntegerSBXCrossover(probability=crossover_proba),
    termination_criterion=StoppingByEvaluations(max_evaluations=max_evals)
)
```

Figure 11. Code snippet of NSGA-II Algorithm Function Call in jMetalPy

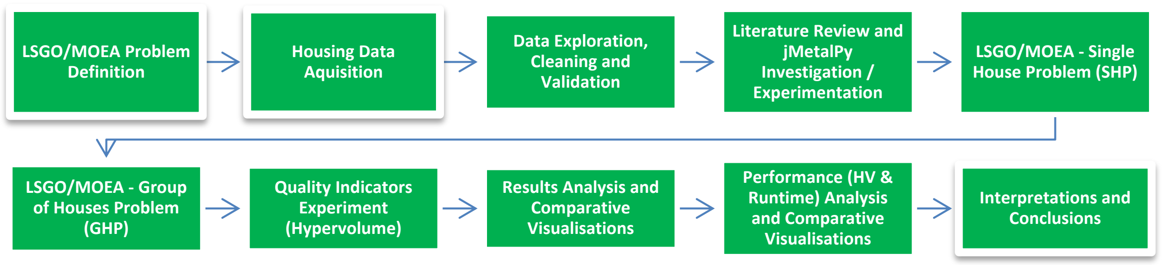
All the variables used in the NSGA-II function call, are picked iteratively from the dictionary of genetic hyperparameter values.

5 Methodology

The choice of methodologies was driven by this project's goals and research approach set out in Section 1.2, Section 1.4 and Chapter 2.

5.1 Project Methodology

Taking cognisance of the introductory considerations set out in Chapter 1 and, of the situational and contextual considerations set out in Chapter 2, a bespoke project methodology was adopted, which was applied iteratively, as outlined in Figure 12 below.



LSGO/MOEA Problem Definition
Housing Data Aquisition
Data Exploration, Cleaning and Validation
Literature Review and jMetalPy Investigation / Experimentation
LSGO/MOEA - Single House Problem (SHP)
LSGO/MOEA - Group of Houses Problem (GHP)
Quality Indicators Experiment (Hypervolume)
Results Analysis and Comparative Visualisations
Performance (HV & Runtime) Analysis and Comparative Visualisations
Interpretations and Conclusions

Figure 12. Outline of Bespoke Project Methodology

5.1.1 Tooling – Software

Jupyter Notebooks were used for their capability to simultaneously (a) support testing whilst developing, (b) facilitate the creation and integration of Python code with equations, computational output and visualisations, interlaced with explanatory text within a single in a single document [40], which can be shared with interested parties e.g.; other researchers. **Jupyter Notebooks** were run on the **Anaconda platform** because it is the recommended distribution [40] for both Python and **Jupyter Notebooks** and is easy to maintain.

Google Colab Pro was chosen for online demonstrations for its on-demand capability. However, due to limitations in tenanted computational capability , including for **Google Colab Pro+** (*whose runtime performance threshold is limited to 11 hours*), its use was restricted to being a repository to share project artefacts and outputs with the academic supervisors.

The following Python libraries, toolkits and frameworks were used;

- **OS** – To perform operating system tasks.
- **Sys** – To interact directly with command line arguments.
- **Time** – To represent and use time functions.
- **Pandas** – To use dataframes for variable data structures, data manipulation and analysis.
- **NumPy** – To use arrays for variable data structures, data manipulation and analysis.
- **Intertools** – To use the mathematical function, *product*.
- **Matplotlib** – To plot graphs and support data visualisation for feedback simplification.
- **jMetalPy** – To develop metaheuristics-based MOEA software for LSGO.

This project additionally used the following supplementary software;

1. **MS Excel** – To perform exploratory data analysis and data visualisation.
2. **MS Word** and **Adobe Acrobat** – To document the dissertation write-up.

5.1.2 Tooling – Hardware

An **Apple iMac** running **MacOS Big Sur 11.6** O/S with the following specifications was used;

- Single 8-core, Intel i9 processor, running at 3.6GHz (*Turbo Boost to 5.0GHz*).
- Radeon Pro 575X 4GB GPU.
- 32GB (*2x16GB*) 2666MHz DDR4 memory.

5.2 Initial Data Exploration and Data Preparation

The initial data dimensions was 27,150 rows and 37 columns, for 935 single houses in 30 separate groups of houses. Of the 37 columns, 10 were target variables for the MOEA objectives, for multi-variable decision making for both SHP and GHP. The dataset had a variable number of rows per single house, with each separate row representing a unique configuration i.e.; different combination of intervention options applied to that house.

It was not possible to ascertain that the data was correctly presented by hand, thereby necessitating the use of data science techniques to check the validity of the dataset and correct any anomalous data prior to execution of the metaheuristics-based MOEA.

The largely superficial anomalies corrected or improved included;

1. New '*dhs*' intervention option and its corresponding '*cost_dhs*' column added.
2. Duplicating each of the 27,150 configurations, so that the new '*dhs*' column could present '*dhs*' intervention as *True* in one configuration and *False* in the additional duplicate configuration.
3. Changed *£* sign to *GBP* because Python treated the *£* sign as an operator rather than as text.
4. Changed column names of the decision-variables for consistency in naming convention; i.e.; change (a) '*biomass heat*' column to '*biom*' and (b) '*Cost_biomass*' column to '*Cost_biom*'.

Post-data cleansing dimensions of the modified data was; 54,300 rows and 38 columns, for 935 single houses in 30 separate groups of houses. Of these 38 columns, 11 were target features for the MOEA objectives supporting multi-variable decision making for both SHP and GHP.

This modified dataset retained the convention of having a unique house configuration per row, each representing a unique combination of intervention options applied to that house.

5.3 Metaheuristics-based Optimisation

The grouping of algorithms that use randomness to find optimal solutions for 'hard problems' is known as *stochastic optimisation*. In turn, *metaheuristics* is the primary sub-field in stochastic optimisation [41]. Metaheuristics are sometimes referred to as '*black box optimisation*' because they generally find solutions with little help from the initiator of the problem simulation.

The LSGO global-problem housing improvements Use Case involves multiple decision-variables, thus creating vast search spaces, which is challenging for many EA's. Therefore, for MOEA software to be developed to satisfactorily make millions of multi-variable decisions, the chosen MOEA would have to satisfy all three characteristics of metaheuristic characteristics [10], [11] i.e.;

1. **Population-Based** – Able to generate better new solutions from current deficient solutions.
2. **Fitness-Oriented** – Able to rank quality of solutions by a fitness value.
3. **Variation-Driven** – Able to vary composition of solutions, to generate better new solutions.

NSGA-II satisfied these characteristics, was chosen and configured in jMetalPy to address the project's goals specified in Section 1.2 and the research questions specified in Section 1.3.

5.3.1 Metaheuristics Principles

Unlike traditional rules-based computation, metaheuristics takes inspiration from nature [6] and the biological principles of evolution by natural selection [42] to solve optimisation problems using EA, as asserted by Zolpakar et.al [43]. Manoharan [44] found that EA that additionally use population-based '*feed forward*' metaheuristics to hasten convergence to obtain more accurate solutions. Hao et.al [45] found that MOEA solve discrete or continuous optimisation problems such as the knapsack problem, using the reference point (RP) as the utopian point.

MOEA use partial search techniques to reduce computational effort but yield good approximations of the results by dynamic application of metaheuristics principles and stochastic methods [46] and as Zhao et.al [25] argued, minimise the potential for getting stuck in *local minima* or *local maxima*.

Chromosomes are *solutions* [47] and the **genes** in chromosomes are *elements in a solution* [47] that are **encoded** in a specific way, using specific patterns aligned to *types of solutions* e.g.; binary encoding, value (*integer*) encoding, permutation encoding or tree encoding. Each gene can **mutate** i.e.; to randomly select chromosomes from which to create new *offspring* i.e.; *children* who make up the next generation of chromosomes, purposely to diversify the **population**. Each gene can **crossover** i.e.; some parts of chromosomes [47] and converge into new chromosomes. Each gene has a **locus** i.e.; position in chromosome; that can alter its *non-dominance* status.

Drezner & Drezner [4] refer to crossover effects as **recombination**, as do Wu et.al [48] who described the effects of mutation and crossover as **genetic shuffling**, with *frog leaping* effect on gene pools.

The MOEA software developed by this project incorporated all concepts outlined in Sub-section 5.3.1.

5.3.2 Encoding/Decoding

To apply MOEA to optimisation problems, an encoding method is required to provide the framework or template for solutions to be organised and computationally processed consistently. Malik [49] outlined these encoding methods as follows;

1. **Binary Encoding** is commonly used due of its simplicity. Each chromosome is a string of *True* or *False* values i.e.; either 0 or 1. SHP used *binary* to show whether any one of the 11 target variables was included in an SHP solution.
2. **Value Encoding** (*or Integer Encoding*) uses real numbers to describe transition state of each gene in a chromosome. Each chromosome is a sequence of values, usually real numbers (*integers*) or objects. GHP used integer values to show state transition status of *one of four types of decisions* associated with any of the 11 target variables in GHP solutions.
3. **Permutation Encoding** is used for ordering problems that have a meaningful sequence e.g.; the travelling salesman problem (TSP), in which each chromosome is a number string representing a specific position in a sequence.
4. **Tree Encoding** is used for evolving programs, in which every chromosome is a tree of objects e.g.; commands or functions. It is similar to decision trees in ML.

5.3.3 Steps for Implementation of Metaheuristics Principles in EA

Figure 5 and Figure 7 show the four key steps [6] of EA metaheuristics, and by extension, MOEA;

1. **Initialisation** – Creation of an initial population of solutions to evaluate.
2. **Selection** – Picking the fitness function to evaluate solutions.
3. **Genetic Operators** – Metaheuristics techniques that strengthen the gene pool i.e.; resultant solutions and subsequent future solutions.
4. **Termination** – Criteria to stop execution of the EA.

5.3.4 Genetic Operators

As described in Sub-section 5.3.1 and in Sub-section 5.3.3, MOEA use genetic operators to create and maintain genetic diversity in the same way as happens in nature, outlined as follows;

5.3.4.1 Replication / Replication Probabilities

Replication starts with randomised selection of the first **chromosomes** to evaluate within a pre-defined encoding strategy, as espoused by Malik [49]. The chromosomes become **parents** of the population. For SHP, binary encoding was used, and parents were chosen using randomness inspired by Zhao et.al [25] and Stanhope et.al [50] as detailed in Section 5.4. For GHP, value (*integer*) encoding was used, with the parents likewise chosen with similar randomness.

Replication reconfigures chromosomes by manipulating parents genes via a combination of ***crossover*** and ***mutation***. This creates stronger offspring with better growth rate than the parents i.e.; solutions with better QI's. This is achieved by limiting replication to the best solutions to create stronger offspring, by excluding weaker offspring from new generations [6]. The full set of solutions is a **genome**, and in the context of this project, **genome** are the full set of SHP or GHP solutions.

The replication process reflects the anatomy of GA's like MOEA as outlined by Gad [12] in Figure 13.

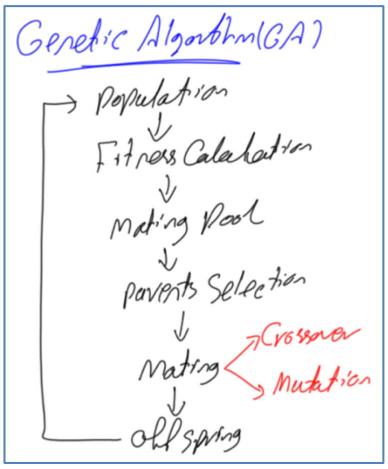


Figure 13. Genetic Algorithm Steps – Process of Population/Chromosome Replication [12]

Evolution occurs after all the steps in the replication process above have been performed [41].

5.3.4.2 Populations

Population is the full set of chromosomes that make up a chromosome [47]. Larger **populations** have more potential for diversity [51] in future populations. To leverage this, this project incrementally doubled the values for '*population*' and '*offspring population*' in the genetic hyperparameters dictionary for NSGA-II from the starting value of 50 up to 800.

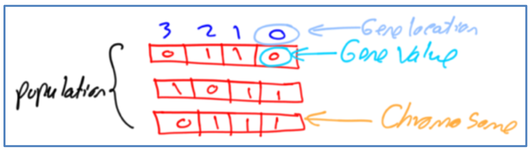


Figure 14. Population with Chromosome and Genes [12]

5.3.4.3 Mutation

Mutation is the process of randomly changing of parts of a solution to increase diversity of a population [47], usually done with uniform mutation probability (*rate*), which sets likelihood that a gene (*element*) in the chromosomes (*solutions*) shall be randomly flipped with a gene from a different solution, e.g.; if a chromosome is encoded as a binary string of length 100 if you have 1% mutation probability, then on average, 1 of 100 bits shall be randomly swapped [52].

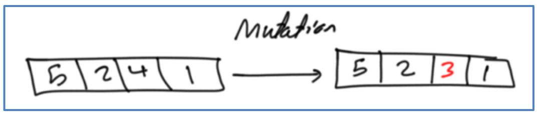


Figure 15. Mutation by Randomly updating some Genes [12]

The example above is *bit-flip mutation* and is just one of a number of different types of mutation.

5.3.4.4 Crossover

Crossover is the process of randomly swapping of part of *solutions* to introduce mixing within solutions in a particular subspace [47], using a uniform probability (*rate*), which sets likelihood that a gene (*elements*) in the chromosomes (*solutions*) shall be randomly swapped with a gene from a different solution, e.g.; if a chromosome is encoded as a binary string of length 100 if you have 90% crossover probability, then on average, 90 of 100 bits shall be randomly swapped [52]. This exemplar for illustrative purposes only, represents just one of a number of different types of crossover, that is commonly referred to as *uniform crossover*. There are other types of crossover e.g.; *1-point crossover* and *2-point crossover*.

Luke [41] noted that randomness in its selection criteria gives **crossover** in GA's the ability to simulate genetic reconfiguration as in nature. **Crossover** can be applied in many ways [52], typically, either *moderately to minimise asymmetric replication*; or *maximised to create asymmetric replication*.

Crossover is typically set to high probability, in the (*0.8..0.95*) range, whereas the opposite is true for **mutation** which flips some digits of an encoded string to generate new solutions, and is typically set to low probability, in the (*0.001 to 0.05*) range [47].

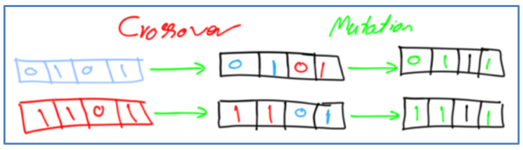


Figure 16. Crossover and Mutation [12]

Crossover and **mutation** are simply the genetic techniques that MOEA's use to make new solutions.

5.3.4.5 Selection

Selection is the process of using the fitness function *f(x)* to pick and evaluate solutions [4], [6]. The elitist selection of the fittest criteria ensures that only the best solutions are propagated [47], by selecting the best solutions. **Crossover** then combines the solutions e.g.; for this project, a *thermal insulation intervention option* from Parent-1, and a *thermal glazing intervention* from Parent-2. **Mutation** then makes a small change to the modified solution e.g.; to *upgrade* or *downgrade* the *thermal insulation intervention option*. In this example, 2 solutions are picked at random, and the best of them is selected. This is one of a number of different selection methods, that is referred to as *tournament selection*, currently the only selection method implemented in jMetalPy for NSGA-II.

Selection determines how MOEAs focus on *exploitation* i.e.; *replicating* known good solutions rather than exploring completely new unknown solutions. As with many hyperparameters, it's not clear ahead of time what their best setting is, hence experimentation to find out.

5.3.4.6 Termination

Maghawry et.al [53] describe **termination** as the set of defined conditions which when met, the EA stops running [6] and evaluation of new generations of **chromosomes** stops. jMetalPy uses its '*max-evaluations*' parameter as the termination variable. It represents the maximum number of generations to evaluate and was deployed as a *constant* for all NSGA-II runs.

Whilst possible, no performance threshold conditions e.g.; maximum runtime set for termination.

5.3.5 NSGA-II Parameters

NSGA-II also uses non-genetic operator parameters as computational parameters for traditional software engineering purposes, that are described here in Sub-section 5.3.5. They set the operational parameters for automated NSGA-II function calls, and in so doing, enable MOEA software to operate LSGO simulations unmanned for long periods of time, uninterrupted.

5.3.5.1 Generations (Maximum Evaluations)

As described in Sub-section 5.3.4, jMetalPy's NSGA-II implementation uses '*max evaluations*' variable as the NSGA-II parameter to enforce the *termination* genetic operator. It is a constant that sets the upper threshold for generations to execute per NSGA-II function call, before picking the best solution.

Given this project's goal of finding factors that contribute to MOEA efficacy, this NSGA-II parameter was treated as a constant for all NSGA-II runs, so as to maintain consistent operating conditions across all NSGA-II runs, and thus helped assess genetic hyperparameters performance and impact.

5.3.5.2 Constraints

The constraints NSGA-II parameters set the **boundaries** in which the NSGA-II algorithm operates. Whist there can be many types of constraints, two are typically such parameters, namely (a) *Lower Bound* and (b) *Upper Bound*.

The lower bound sets the *lowest values* for the choice of encoding must use for the fitness function ***f(x)***, and conversely, the upper bound sets the *highest value* that can be used. These values are used to calculate the largest (*i.e.; maximum possible*) search space size.

5.4 NSGA-II Computation with Genetic Hyperparameters

Unlike conventional ML based hyperparameters that follow the conventional linear or random patterns as illustrated by Stalfort [54], metaheuristics [42] take inspiration from natural evolution [6] to find good solutions and to subsequently propagate even better solutions in future generations. Additionally, metaheuristics are themselves algorithms with hyperparameters to control their behaviour.

Given the limited research to benchmark on separability in LSGO, initial genetic hyperparameter values used were initially benchmarked from jMetalPy examples in the GitHub repository and online documentation. Genetic hyperparameter values were added using genetic hyperparametric rule of thumb principles by researchers [54], [51], [50] [10] and [11] that presented the consistent view that;

1. Expect performance variations with identical genetic hyperparameter combinations operating with different encoding techniques e.g.; use smaller mutation/crossover probabilities for binary encoding.
2. The smaller the population, the quicker the convergence, with a potential downside of being trapped in *local minima*. And conversely, the higher the population, the slower the convergence, with the potential downside of being stuck in local maxima.

As a result, some form of sensitivity analysis would be required, with genetic hyperparameters tuning implemented guided by the genetic hyperparametric inter-relationships principles presented by Wang et.al [51], and from which a dictionary of possible values was created for this purpose.

NSGA-II MOEAs were tuned by running all variations of the maximum possible number of genetic hyperparameters configurations, as espoused in prior research initially by [50] and later by Eiben et.al. [10] and [11], enabling analysis of the performance of various NSGA-II configurations.

5.4.1 Genetic Hyperparameter Values Selection Strategy

The selection of optimum genetic hyperparameter values was influenced by Stanhope & Daida [50]. For computational resource optimisation purposes, a maximum of 4 genetic hyperparameter values per genetic operator parameter was the limit chosen for inclusion in a Python dictionary structure.

Selection of genetic hyperparameter values was based on targeted guidance inspired by Zhao et.al [25], necessitating the need minimise testing turnaround time. As a result, an initial populations size (*including offspring populations*) of 50 was chosen. Taking guidance from the findings from Wang et.al's [51] research, subsequent values for population size would be doubled for up to 3 additional values, to a limit of 4 values per genetic hyperparameter. Due to the aforementioned limitations in computational power, this was therefore limited to 64 configurations of (a) proportionately spaced values specified in a dictionary, including with the (0.001, 0.05) range for Crossover and (0.8, 0.95) range for Mutation.

The same principle was applied to the NSGA-II parameters specified in Section 5.3.5.

5.5 Performance Metrics and Measurement

In the Sub-sections below, the combination of non-functional and functional performance measures used to assess search space optimisation, NSGA-II efficacy and performance thresholds e.g.; maximum evaluations or runtime boundaries, are outlined.

5.5.1 Fitness Scores

Fitness functions ***f(x)*** also referred to as evaluation functions, iteratively take solutions being evaluated as inputs, as outlined in multiple stages of optimisation in Figure 13; to produce fitness scores as output, that show good that solution is [55], relative to MOEA objectives. Fitness functions evaluate the proximity of a solution to the optimum solution [56]. For optimisation problems, it is usually the sum of calculated x-axis and y-axis parameters relative to the problem domain, which should provide predictable results i.e.; the best solutions should have the best fitness scores [56] and vice-versa.

For this project, fitness functions ***f(x)*** assessed the quality of solutions against objective functions, i.e.; cost of thermal efficiency interventions vs reductions in energy consumption. In addition, they assessed MOEA efficacy. The fitness score outputs were stored in the respective '*PF Index*' columns for both SHP and GHP, and HV fitness scores were stored in '*IndicatorValue*' and 'HV *Ranking*' columns.

Gad [12], found that higher fitness scores reflected higher quality solution, and the converse are cost function scores, also referred to as loss function scores. jMetalPy's QI experiments typically use an indicator function, with normalised scores in the (*0, 10*) range, and which is why normalisation is supposedly required for credible QI results, when non-normalised scores can also be obtained.

This project used an incremental ranking series as an index of solution quality, stored in a '*PF Index*' column of results data, which was used to store fitness score rankings of solutions, relative to their objective functions i.e.; by Cost vs Energy. A similar approach was used for MOEA efficacy, with ranked PF quality fitness scores stored in an '*HV Ranking*' column for the QI experimentation results.

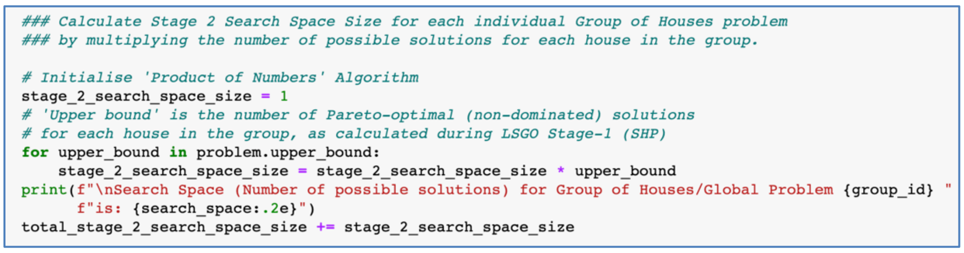
All of this was in accordance with the goals set out in Section 1.2, Section 4.3 and Section 7.1.

5.5.2 Search Space Metrics and Measurement

Brownlee et.al's [3] search space decomposition that was used by this project is conceptualised in Sub-section 4.1.2. Search space is traditionally the product of all possible solutions. MOEA efficacy can be measured in terms of the search space size vs a non-functional performance measure e.g.; runtime or HV, or computational resources utilisation vs search space size.

In linearity, the larger the search space, the more computational effort is required to solve problems, however, with metaheuristics, there is significant potential for this to be significantly reduced due to its inherent self-learning properties.

For this project, search space size was calculated as product of all the upper bound limits for each GHP solutions constituent elements i.e.; one of 31 groups of houses a group of houses, calculated as shown in the code snippet in Figure 17 below.



```
### Calculate Stage 2 Search Space Size for each individual Group of Houses problem
### by multiplying the number of possible solutions for each house in the group.

# Initialise 'Product of Numbers' Algorithm
stage_2_search_space_size = 1
# 'Upper bound' is the number of Pareto-optimal (non-dominated) solutions
# for each house in the group, as calculated during LSGO Stage-1 (SHP)
for upper_bound in problem.upper_bound:
    stage_2_search_space_size = stage_2_search_space_size * upper_bound
print(f"\nSearch Space (Number of possible solutions) for Group of Houses/Global Problem {group_id} "
      f"is: {search_space:.2e}")
total_stage_2_search_space_size += stage_2_search_space_size
```

Figure 17. Stage-2 (GHP) Search Space Size Calculation Code Snippet

Search space size values were shown in output cells in scientific notation alongside runtime. Both were then added to the GHP Results CSV output file, and also plotted in visualisations, a sample of which can be seen in Sub-section 7.2.8.

Scientific notation was used because of the very large numbers involved, take for example;

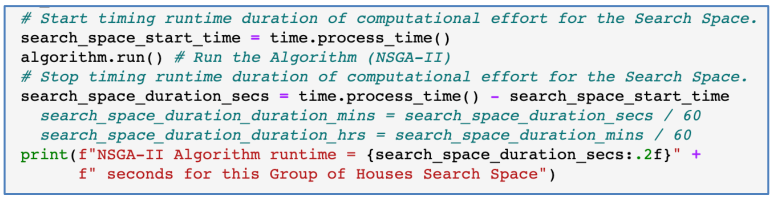
- 1.094e+02 = 1.094*10^2 = 1.094*100 = 109.4
- Therefore, 5.00e+31 = 5.00*10^31 = A very big and long number ending with 31 zero's.

5.5.3 Runtime Metrics and Measurement

MOEA efficacy can be measured in terms of runtime vs search space size for each individual NSGA-II function call. It can be a useful measure of computational resources utilisation, especially if done in a clean room environment using a fit for purpose tool e.g.; Intel's pyRAPL that can measure;

- CPU utilisation
- DRAM utilisation
- Runtime.

For this project, runtime is an inverse relationship between search space size, algorithm complexity and computational power. It was calculated for each individual NSGA-II function call for the defined sub-problems i.e.; SHP or GHP, and was implemented as shown in code snippet in Figure 18 below.



```
# Start timing runtime duration of computational effort for the Search Space.
search_space_start_time = time.process_time()
algorithm.run() # Run the Algorithm (NSGA-II)
# Stop timing runtime duration of computational effort for the Search Space.
search_space_duration_secs = time.process_time() - search_space_start_time
  search_space_duration_duration_mins = search_space_duration_secs / 60
  search_space_duration_duration_hrs = search_space_duration_mins / 60
print(f"NSGA-II Algorithm runtime = {search_space_duration_secs:.2f}" +
      f" seconds for this Group of Houses Search Space")
```

Figure 18. NSGA-II (GHP) Runtime Calculation Code Snippet

Runtime results are displayed in output cells besides search space size, and both are added to the GHP Results CSV. A sample of visualisations are created can be seen in Sub-section 7.2.8.

It is arguable that that runtime calculations are unlikely to be accurate because the computer was not used exclusively for NSGA-II function calls. So, runtime calculations used the *time.process_time()* Python method which measures processor utilisation, a more accurate measure. A recommendation for future work was also added to revisit this using pyRAPL.

5.6 Quality Indicators Metrics and Measurement (jMetalPy)

Quality indicators (QI) are statistical measures that were used for statistical significance testing. QI are similar to scoring metrics in ML. QI can be used for EA efficacy comparisons and were used by this project to assess NSGA-II efficacy, proving that this can be done within jMetalPy, to assess efficacy of discrete genetic hyperparameter configurations.

jMetalPy has multiple QI, including; Hypervolume (HV) aka *Lebesgue measure or S-metric* [57], Generational Distance (GD) and Epsilon Indicators (EI). However, only used HV for GHP, because GD and EI are only used in NSGA-II for synthetic benchmarks, whereas this project deals with a real-world LSGO.

Notwithstanding, the use of additional QI not used by this project has been ascribed to Future Work.

5.6.1 Hypervolume

Russo et.al [58] and Cao et.al [59] describe HV as the area (*or volume*) between solutions on the PF and the RP. It is normally the maximum known value for each MOEA objective i.e.; the *global maxima*. The converse of the RP is the Nadir Point, which is the *global minima* [60]. HV is also a polytope between the MOEAs PF and the Nadir Point [60]. These points are used to measure PF optimality [38] and EA efficacy, by determining proportion of search space that the PF *weakly dominates* [60], to obtain HV score.

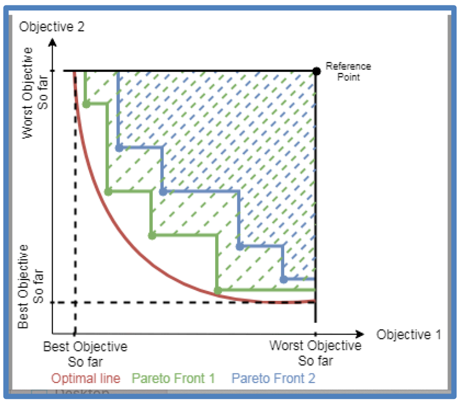
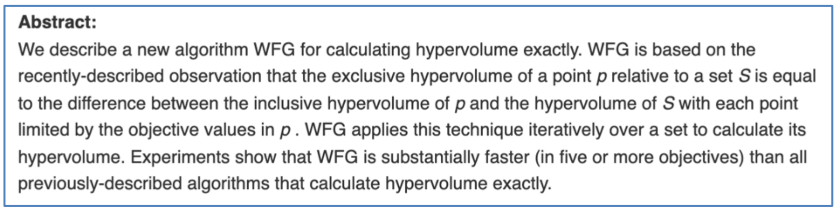


Figure 19. HV Quality Indicator Illustration [61]

For this project, the RP was the maximum possible value for the MOEA objective functions i.e.; *max_energy* and *max_cost*. HV values are normally normalised in the (0.0, 1.0) range, and the larger the HV value, the better the algorithm. Therefore, the closer GHP PF's are to the GHP PF's x and y axes, the better the MOEA (*distinguished by its genetic hyperparameter configuration*) that created the PF. That is the desired outcome. For MOEAs, the performance goal is always to maximize the HV [60]. When normalised, the closer the HV is to 1.0 the better the that created the PF.

Whilst there are a number of HV measures in the literature, it was not possible to find out the specific HV measure used by jMetalPy. Such more granular measures include; (a) Gradient of Hypervolume, (b) Hypervolume contribution, (c) S-metric-based Expected Improvement (SExI) and (d) Online Convergence Detection (*OCD-Classic and OCD-HV*). jMetalPy's HV calculation is black box, however, well known approaches includes While et.al's [62] algorithm described in the *Abstract* of their paper as shown in Figure 20 below.



Abstract:

We describe a new algorithm WFG for calculating hypervolume exactly. WFG is based on the recently-described observation that the exclusive hypervolume of a point *p* relative to a set *S* is equal to the difference between the inclusive hypervolume of *p* and the hypervolume of *S* with each point limited by the objective values in *p* . WFG applies this technique iteratively over a set to calculate its hypervolume. Experiments show that WFG is substantially faster (in five or more objectives) than all previously-described algorithms that calculate hypervolume exactly.

Figure 20. Algorithm to Calculate HV [62]

HV results were obtained by running jMetalPy's QI's experiment, and the HV results were saved to the '*Quality Indicators Results*' CSV output file, and were plotted in GHP PF visualisations with HV added as a shaded area above the curve, samples of which can be seen in Sub-section 7.1.10.

5.6.2 Experimentation Methodology

jMetalPy has in-built functionality to perform QI's experiments, done using the following steps:

1. **What shall be compared?** For this project it was NSGA-II configurations. to be compared and the benchmark problems to be used.
2. **What statistical tests shall be conducted?** Apply HV QI's statistical significance tests to each separate GHP PF.
3. **What shall be collated/stored?** Collate and store the PF's from each NSGA-II run per GHP evaluated, with each PF a separate job for the experiment. Also store the HV results.
4. **What shall the results be presented?** Calculate MOEA (NSGA-II) efficacy by HV ranking;
    a. per unique configuration.
    b. per unique configuration for each individual GHP PF.
    c. Mean HV for unique configuration used for each of 30 separate *extrema* GHP PF's.

5.6.3 QI Experimentation in jMetalPy

HV comparisons used principles espoused by Russo et.al [58] and Cao et.al [59] for choosing RP's for two-dimensional PF's to calculate the HV for NSGA-II.

To give equal weight to all the MOEA objectives in assessments of MOEA efficacy, Benítez-Hidalgo et.al [38] recommended statistically significant tests for meaningful comparison of genetic hyperparameter configurations, and by extension, HV QI vis-à-vis MOEA objectives i.e.; ***min*** to ***max energy*** vs ***min*** to ***max cost***. It is recommended that this range of values are normalised prior to QI experiment. Time constraints however, limited a full study of how to extend the QI experiment super-class and API and normalise the values i.e.; ***min*** to ***max energy*** vs ***min*** to ***max cost***; prior to running QI experiment.

So, to create HV results for simplified feedback, QI experiments were instead performed iteratively per NSGA-II function call, per GHP PF, using non-normalised MOEA objective functions real scores. Results were presented in tabular format and were used to create new GHP PF visualisations with HV as a *shaded area above the curve of the GHP PF line*, relative to the RP.

Subsequent to the QI experiments, the tabular PF values and HV results (*real values*) were appended to the *QualityIndicators.csv* output file as shown in Figure 21 below, and also as GHP PF visualisations with HV shown relative to the RP.

$$\boxed{X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}}$$

Figure 21. Formula used to Normalise the Axes for HV purposes

Normalising the HV scores and PF values was done after QI experiments, using the formula in Figure 21 above. To test for normalisation process integrity, the *before* and *after* normalisation GHP PF's HV values were compared (*statistically and visually*), for any skewedness in proportionality, but none was found.

QI experiments were conducted according to steps proposed by Benítez-Hidalgo et.al [38]. Results were extracted into Python variables and then saved externally into CSV's.

6 Software Development using MOEA

As introduced in Section 1.1 and Section 1.2, this project's purpose and goal was to extend an existing MOEA and create Python software using jMetalPy for LSGO global-problem to optimise thermal efficiency retrofit of public housing stock. A key factor in this regard, was to incorporate *separability* and *explainability*, whilst simultaneously addressing the research questions posed in Section 1.3.

Explainability was integral to software development, by integrating causal inference and mapping of algorithm steps (*e.g.; genetic hyperparameter configurations* and *constitution of GHP solutions*), as inputs; to specific measurable outcomes (*e.g.; PF's, HV* and *runtime*), as outputs, including assessing the significance of these discrete inputs in satisfying MOEA objectives including MOEA efficacy as measured by HV, and in the quality of automated multi-variable decision making as measured by LSGO targets to minimise energy consumption and costs of interventions.

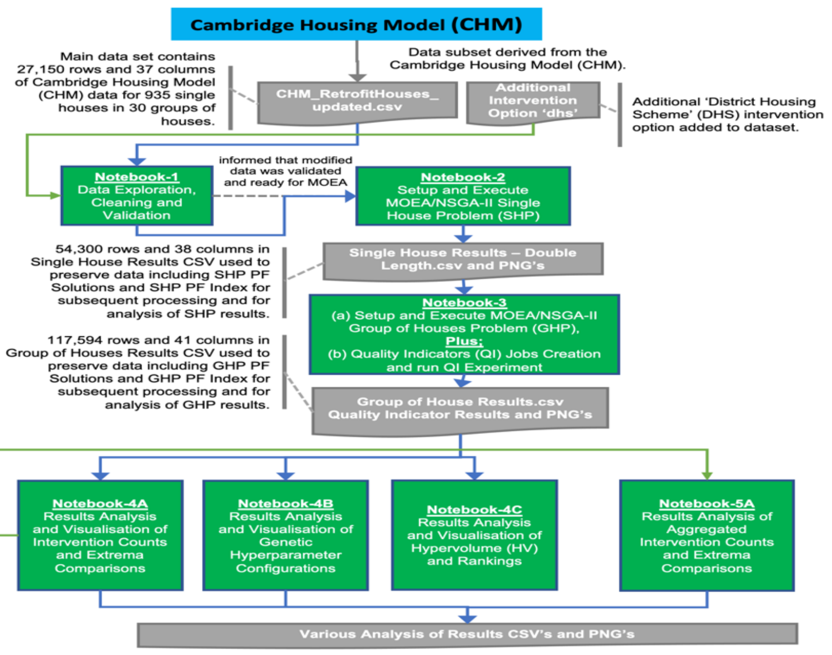
Using *causal inference* was a foundation for results analysis, particularly in determining constituent thermal efficiency intervention options in solutions, influenced the millions of *decision-variables*, i.e.; towards understanding which variables are important in driving the MOEA objectives.

Considering non-functional performance, earlier researchers work was noted e.g.; Cohoon et.al [63] finding that EA's introduce computational complexity, Ye et.al [64] and Maier et.al [65], finding that EA execution in turn requires significant computational resource utilisation due to EA processing to the last defined generation *(max_evaluations operator for NSGA-II*). Cohoon et.al [63] additionally found that when it comes to EA's, "*there may be no direct link between algorithm complexity and problem complexity*".

Taking cognisance of these considerations extensive runtime requirements, this project modularised software development and created bite size components i.e.; seven (7) separate Notebooks that could be processed end to end on most researchers PC's, each performing a discrete set of tasks.

6.1 Development Environment

Jupyter Notebooks running on Anaconda platform were used for the reasons [40] explained in Sub-section 5.1.1, and the development environment was organised as outlined in Figure 22 below;



Cambridge Housing Model (CHM)
Main data set contains 27,150 rows and 37 columns of Cambridge Housing Model (CHM) data for 935 single houses in 30 groups of houses.
Data subset derived from the Cambridge Housing Model (CHM).
CHM_RetrofitHouses_updated.csv
Additional Intervention Option 'dhs'
Additional 'District Housing Scheme' (DHS) intervention option added to dataset.
Notebook-1 Data Exploration, Cleaning and Validation
informed that modified data was validated and ready for MOEA
Notebook-2 Setup and Execute MOEA/NSGA-II Single House Problem (SHP)
54,300 rows and 38 columns in Single House Results CSV used to preserve data including SHP PF Solutions and SHP PF Index for subsequent processing and for analysis of SHP results.
Single House Results – Double Length.csv and PNG's
Notebook-3 (a) Setup and Execute MOEA/NSGA-II Group of Houses Problem (GHP), Plus; (b) Quality Indicators (QI) Jobs Creation and run QI Experiment
117,594 rows and 41 columns in Group of Houses Results CSV used to preserve data including GHP PF Solutions and GHP PF Index for subsequent processing and for analysis of GHP results.
Group of House Results.csv Quality Indicator Results and PNG's
Notebook-4A Results Analysis and Visualisation of Intervention Counts and Extrema Comparisons
Notebook-4B Results Analysis and Visualisation of Genetic Hyperparameter Configurations
Notebook-4C Results Analysis and Visualisation of Hypervolume (HV) and Rankings
Notebook-5A Results Analysis of Aggregated Intervention Counts and Extrema Comparisons
Various Analysis of Results CSV's and PNG's

Figure 22. **e2e Project Computation (*Jupyter Notebooks*) and Data Flowchart**

The development environment took advantage of the multi-threading capability in the choice of hardware outlined in Sub-section 5.1.1 and software outlined in Sub-section 5.1.2.

6.2 Algorithmic Design

Algorithmic design was crucial, as described in Section 1.2, Section 2.3 and Section 3.1, including;

1. Using multiple design approaches to address different project considerations [63], [65], [66].
2. Optimising expected utilisation of considerable computational resources [63], [65].
3. Creating trust in the MOEA software developed, as espoused by Chawla [67], via combination of deploying (a) '*divide and conquer*' [66] algorithm design techniques, (b) organisational competence and transparency in the code through (c) problem decomposition using separability, (d) causal inference for explainability and (e) maintainability.
4. Using appropriate data structures to explore, manipulate, visualise, store data in-flight and externally; and retrieve data [66] in jMetalPy and Python, to optimise computational resources, particularly memory, CPU utilisation and runtime.

Figure 23 shows bespoke algorithm design pseudocode for the SHP sub-problem.

```
Algorithm-1 Single House Sub-Problem (SHP) Algorithm
1: INPUT updated CHM data sub-set → load into Pandas dataframe;
2: explore and validate the data → data validation;
3: calculate the number of houses in CHM dataset, by finding their start and end positions in the dataframe;
4: SHP decomposition, by extending jMetalPy Binary Problem super class, abstract classes, and abstract methods;
5: for i (number of Houses) ← 1 to 935 do
6:     define SHP for single house
7:     run NSGA-II algorithm with baseline metaheuristics configuration
8:     collate results (SHP Pareto Front of non-dominated solutions)
9:     create results visualisations (SHP Pareto Front of non-dominated solutions)
10:    save results visualisations to PNG (SHP Pareto Front of non-dominated solutions)
11:    calculate solution ranking (SHP PF Index)
12:    update list of overall solution ranking (Overall SHP PF Index) and overall solutions (Overall SHP PF Solutions)
13:    add populated SHP PF Index and SHP PF Solutions columns to the dataframe
14: end for
15: save results dataframe to CSV
```

Figure 23. ALG-1 – Single House Problem (SHP) Pseudocode

Figure 24 shows bespoke algorithm design pseudocode for the GHP problem.

```
Algorithm-2 Group of Houses Global-Problem (GHP) Algorithm
1: INPUT Single House Sub-Problem (SHP) Results data → load into Pandas dataframe;
2: calculate the number of groups in SHP Results data, by finding their start and end positions in the dataframe;
3: GHP decomposition, by extending jMetalPy Integer Problem super class, abstract classes, and abstract methods;
4: create dictionary of genetic hyperparameters → metaheuristics combinations;
5: create results output CSV file → with target column names, including GHP PF Index and GHP PF Solutions
6: for i (number of Runs) ← 1 to 2 do
7:     for j (number of genetic hyperparameter configurations) ← 1 to 64 do
8:         for k (number of Groups of Houses) ← 1 to 30 do
9:             define GHP for a group of houses
10:            run NSGA-II algorithm for current Group of Houses, with currently specified
                   genetic hyperparameters configuration
11:            create a Quality Indicators job for NSGA-II preceding execution
12:            append a Quality Indicators job to a global list of Quality Indicators job to be performed later
13:            collate results (GHP Pareto Front of non-dominated solutions)
14:            create results visualisations (GHP Pareto Front of non-dominated solutions)
15:            save results visualisations to PNG (GHP Pareto Front of non-dominated solutions)
16:            save results tabular data to CSV (all columns of CSV)
17:        end for
18:    end for
19: end for
20: save results dataframe to CSV
```

Figure 24. ALG-2 – Group of Houses Problem (SHP) Pseudocode

6.2.1 Algorithmic Design for Results Analysis

The bespoke algorithm designs for causal inference purposes i.e.; results extraction, analysis and visualisations preparation, follow the similar pattern of loops and code branches outlined in Section 6.2.

Algorithmic design for the steps after MOEA function calls, focused on deploying causal inference by continuous results extraction, analysis, visualisations preparation, deriving findings and insights to inform the conclusions; storing outputs in CSV's and PNG's, benchmarking extrema GHP solutions, use mathematical and statistical mining techniques to collate, analyse and save;

1. Baseline intervention counts and their comparisons.
2. Baseline genetic hyperparameter configurations performance and their comparisons.
3. Aggregated intervention counts and their comparisons.
4. MOEA efficacy through (a) HV comparisons and rankings, and (b) runtime comparisons.

6.2.2 Future Proofing

Best practises implemented to future proof and enable subsequent sub-problems and future researchers to use them the MOEA and supplementary software were;

1. Using PEP-8 Python coding standards to create readable code that is easy to maintain.
2. Creating variables as computed values rather than static hard coded values. Changes in pre-set operating dynamically affect or change the values for the variables.
3. Creating additional columns and rows in SHP outputs for new intervention options i.e.; for '*dhs*' and '*dhs cost*', so that 50% of SHP configurations had '*dhs*' added to their SHP solutions whilst the other 50% did not, as well as '*PF Index*' and '*PF Solutions*' columns.
4. Creating additional columns and rows in GHP outputs, so that subsequent sub-problems and future researchers can make use of them, e.g.; '*PF Index*', '*PF Solutions*', '*house-id*' and an '*Intervention_count*' column for each of the 11 interventions.
5. Automated preparation of results analysis visualisations that can expeditiously inform the findings or alert to the possible need to re-design aspects of the coding.
6. Used *relative path statements* in Notebooks rather than local paths, so they could run anywhere e.g.; other integrated development environments (IDE), Google Colab and on multiple O/S without making changes to path statements, including addition of a *root directory*.
7. Automated creation and storage of results, so that simulation can be done once and analysis subsequently performed on demand, using the save results data.

6.3 MOEA Development in jMetalPy

The systematic activities undertaken to get familiarised with jMetalPy were;

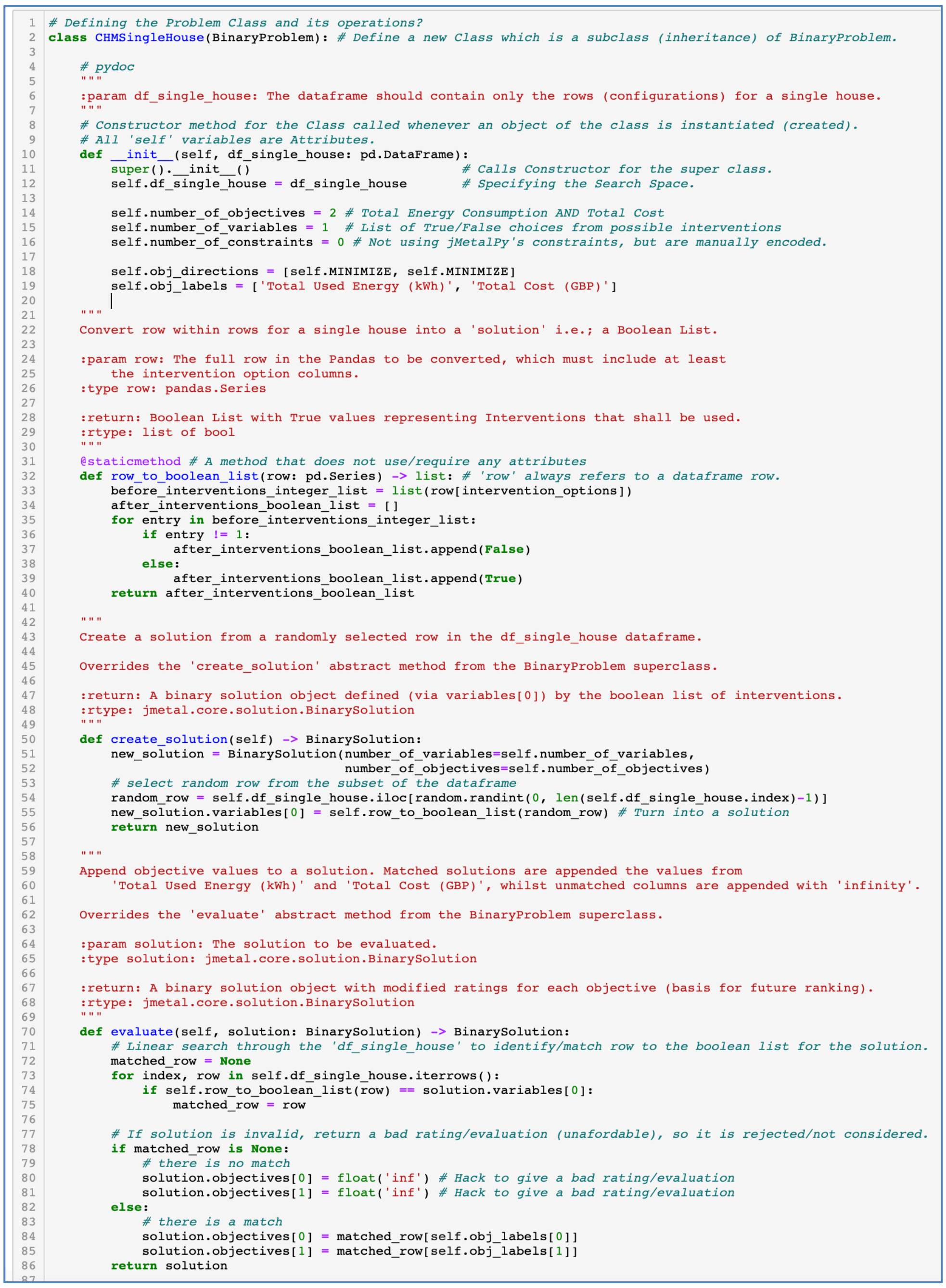
1. Reading scholarly articles which reference jMetalPy including Nebro's [37] benchmark paper that first introduced jMetalPy to the world, as the Python version of jMetal.
2. Study the jMetalPy GitHub repository and online documentation to understand how they are organised so as to be able to make best of use them subsequently. This additionally included studying code examples and replicating the approaches with sample problems and test data.
3. Consulting online communities e.g.; Stack Overflow to address issues encountered during the development of new jMetalPy abstract sub-class extensions to the jMetalPy source code.

6.3.1 jMetalPy Class Inheritance and Abstract sub-classes

Algorithmic design style used for jMetalPy source code and examples of code interactions with its application programming interface (API) were adopted. They are available in jMetalPy GitHub repository and online documentation, and were used as templates for the MOEA software that was developed.

jMetalPy source course is written using extensible constructor super-classes with embedded abstract sub-classes that cannot be instantiated by themselves but are instantiated by the super-class to which they belong, as espoused by Krasnov [68]. The same convention was used to best align with jMetalPy and make it easier for future researchers to use the work created by this project.

Code of extended constructor super-class for SHP *BinaryProblem* class is shown in Figure 25.



```python
# Defining the Problem Class and its operations?
class CHMSingleHouse(BinaryProblem): # Define a new Class which is a subclass (inheritance) of BinaryProblem.

    # pydoc
    """
    :param df_single_house: The dataframe should contain only the rows (configurations) for a single house.
    """
    # Constructor method for the Class called whenever an object of the class is instantiated (created).
    # All 'self' variables are Attributes.
    def __init__(self, df_single_house: pd.DataFrame):
        super().__init__()                          # Calls Constructor for the super class.
        self.df_single_house = df_single_house      # Specifying the Search Space.

        self.number_of_objectives = 2 # Total Energy Consumption AND Total Cost
        self.number_of_variables = 1  # List of True/False choices from possible interventions
        self.number_of_constraints = 0 # Not using jMetalPy's constraints, but are manually encoded.

        self.obj_directions = [self.MINIMIZE, self.MINIMIZE]
        self.obj_labels = ['Total Used Energy (kWh)', 'Total Cost (GBP)']
        |
    """
    Convert row within rows for a single house into a 'solution' i.e.; a Boolean List.

    :param row: The full row in the Pandas to be converted, which must include at least
        the intervention option columns.
    :type row: pandas.Series

    :return: Boolean List with True values representing Interventions that shall be used.
    :rtype: list of bool
    """
    @staticmethod # A method that does not use/require any attributes
    def row_to_boolean_list(row: pd.Series) -> list: # 'row' always refers to a dataframe row.
        before_interventions_integer_list = list(row[intervention_options])
        after_interventions_boolean_list = []
        for entry in before_interventions_integer_list:
            if entry != 1:
                after_interventions_boolean_list.append(False)
            else:
                after_interventions_boolean_list.append(True)
        return after_interventions_boolean_list

    """
    Create a solution from a randomly selected row in the df_single_house dataframe.

    Overrides the 'create_solution' abstract method from the BinaryProblem superclass.

    :return: A binary solution object defined (via variables[0]) by the boolean list of interventions.
    :rtype: jmetal.core.solution.BinarySolution
    """
    def create_solution(self) -> BinarySolution:
        new_solution = BinarySolution(number_of_variables=self.number_of_variables,
                                      number_of_objectives=self.number_of_objectives)
        # select random row from the subset of the dataframe
        random_row = self.df_single_house.iloc[random.randint(0, len(self.df_single_house.index)-1)]
        new_solution.variables[0] = self.row_to_boolean_list(random_row) # Turn into a solution
        return new_solution

    """
    Append objective values to a solution. Matched solutions are appended the values from
        'Total Used Energy (kWh)' and 'Total Cost (GBP)', whilst unmatched columns are appended with 'infinity'.

    Overrides the 'evaluate' abstract method from the BinaryProblem superclass.

    :param solution: The solution to be evaluated.
    :type solution: jmetal.core.solution.BinarySolution

    :return: A binary solution object with modified ratings for each objective (basis for future ranking).
    :rtype: jmetal.core.solution.BinarySolution
    """
    def evaluate(self, solution: BinarySolution) -> BinarySolution:
        # Linear search through the 'df_single_house' to identify/match row to the boolean list for the solution.
        matched_row = None
        for index, row in self.df_single_house.iterrows():
            if self.row_to_boolean_list(row) == solution.variables[0]:
                matched_row = row

        # If solution is invalid, return a bad rating/evaluation (unafordable), so it is rejected/not considered.
        if matched_row is None:
            # there is no match
            solution.objectives[0] = float('inf') # Hack to give a bad rating/evaluation
            solution.objectives[1] = float('inf') # Hack to give a bad rating/evaluation
        else:
            # there is a match
            solution.objectives[0] = matched_row[self.obj_labels[0]]
            solution.objectives[1] = matched_row[self.obj_labels[1]]
        return solution
```

Figure 25. jMetalPy code snippet for SHP Sub-Problem Decomposition

7 Results and Discussions

The approach to the creation of all results output stored externally to the Notebooks, in order to help to address this project's goals set out in Section 1.2 and the research questions in Section 1.3.

Results in tabular format enabled subsequent review of any individual house or any group of houses on the PF's, optimised with discrete MOEA genetic hyperparameters.

PF visualisations enabled visual analysis of any individual house or any group of houses on the PF, optimised with discrete MOEA genetic hyperparameters.

All of the *tabular output files* and *visualisation output files* were used by this project to derive insights, findings, conclusions make recommendations, and can also be used by other researchers, e.g.; for housing policy making, statistical modelling or even for computational simulation purposes.

With the exception of SHP PF's, no visualisations were prepared for SHP, as they were of secondary interest. The principles for creation of visualisations described in Sub-subsection 7.1.5.1 were used.

7.1 Results

7.1.1 SHP Tabular Results

From all SHP NSGA-II function calls, a single tabular file was created with modified data dimensions of; 54,300 rows and 40 columns, for 935 single houses in 30 separate groups of houses.

Of the 40 columns in is output file, 11 were target variables for the MOEA objectives, used for multi-variable decision making for both SHP and GHP. The modified dataset retained the same variable number of rows per single house, with each row representing a unique configuration i.e.; combination of intervention options for that house. Two columns were added to the output file namely;

- PF Index – To hold the rankings of the solutions on the SHP PF.
- PF Solutions – To hold the specific interventions applied to a single house.

The results were extracted in the manner outlined in the pseudocode for *bespoke algorithm-1* presented in Figure 23 in Section 6.2. The output file is named; Single House Results – Double Length.CSV, and a copy is available in this project's artefacts in the project repository.

This file was used as the input file for Notebook-3 (*LSGO for GHP*).

7.1.2 GHP Tabular Results

From all GHP NSGA-II function calls, a single tabular file was created with data dimensions of; 117,594 rows and 40 columns, for 935 single houses in 30 separate groups of houses.

Of the 40 columns in this output file, 11 were target variables for the MOEA objectives, used for multi-variable decision making for GHP only. This GHP modified dataset retained the same variable number of rows per single house, with each row representing a unique configuration i.e.; combination of intervention options for that house. Two columns were added to the output file namely;

- Group PF Index – To hold the rankings of the solutions on the GHP PF.
- Group PF Solutions – To hold the specific interventions applied to a single house.

The results were extracted in the manner outlined in the pseudocode for *bespoke algorithm-2* presented in Figure 24 in Section 6.2. The output file is named; Group of Houses Results.CSV, and a copy is available in this project's artefacts in the project repository.

This file was used as the input file for Notebook-4A (*Results Analysis – Intervention Counts*), and Notebook-5A (*Results Analysis – Aggregated Intervention Counts*).

7.1.3 Results Analysis – Intervention Counts Tabulation

Subsequent to all SHP and GHP NSGA-II function calls, intervention counts were tabulated for the points of interest which are the ***extrema*** on the PF (i.e.; the ***minima***, ***knee-point*** and ***maxima)***.

In effect, this was tabulation of the *lowest (**minima**)*; *best trade-off's (**knee point**)* and *highest (**maxima**)* number of interventions applied across all 935 single houses, for all 64 unique genetic hyperparameter configurations; from which a single tabular output file was created with data dimensions of; 385 rows and 29 columns, for 935 single houses in 30 separate groups of houses.

Of the 40 columns in this output file, 1 was for the *Run number*, 1 was for the *genetic hyperparameter configuration ID*, 1 was for the *number of groups*, another 1 each for the non-constant genetic hyperparameters i.e.; *population (which is also used for offspring population)*, *Mutation* and *Crossover*;11 were for *aggregated interventions values* for each of the individual intervention options; and another 11 were for *aggregated interventions squares* for each of the individual intervention options.

The results were extracted in the manner outlined in the pseudocode for *bespoke algorithm-2* presented in Figure 24 in Section 6.2. The output file is named; Aggregated Interventions Counts.csv; and is available in this project's artefacts in the project repository. It was used as the input file for Notebook-5A.

7.1.4 QI Tabular Results

jMetalPy's QI Experiments functionality automatically creates a number of tabular results when the QI experiment is run, as described in Appendix 2. The most crucial of these output files is the output file is named; QualityIndicators.csv; because its contents are used to derive HV Rankings. All the files created by QI Experiments are available in this project's artefacts in the project repository.

7.1.5 SHP and GHP PF Visualisations

For both SHP and GHP, a visualisation was prepared of each PF of the viable (*non-dominated*) solutions was prepared using both jMetalPy and bespoke Python code developed in conjunction with inbuilt Matplotlib functionality.

A sample of each of the resultant SHP and GHP PF visualisations are shown below;

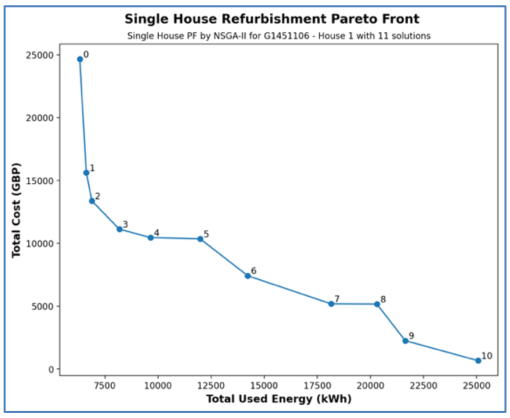


Figure 26. PF for SHP

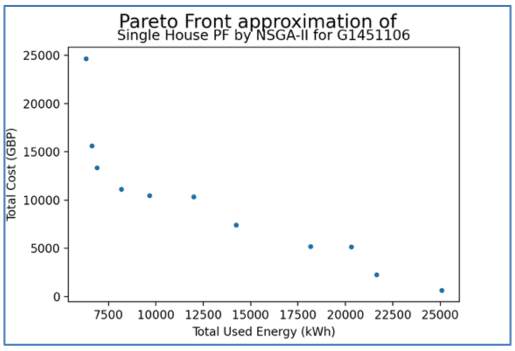


Figure 27. jMetalPy PF for SHP

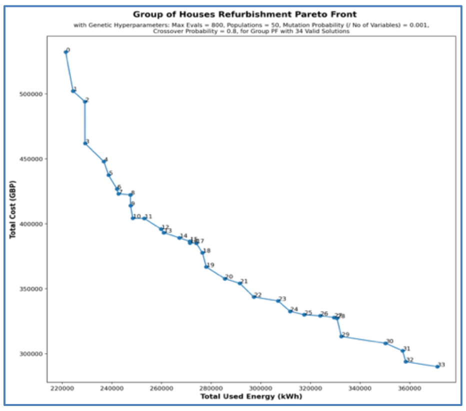


Figure 28. PF for GHP

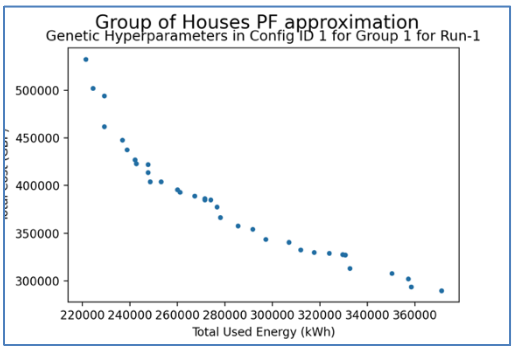


Figure 29. jMetalPy PF for GHP

7.1.5.1 Algorithmic Principles for Creation of Visualisations

To ensure that each PF visualisation could be subsequently reviewed, the number of graphs created in both jMetalPy using bespoke Python code was based on the number of configurations and runs performed, as per the illustration by the equations below;

- Number of single house graphs = ***s*** x ***h***, where = ***s*** is the number of valid solutions per house, and ***h*** is the number of houses.
- Number of group of houses graphs = ***gs*** x ***g*** x ***c*** x ***r***, where = ***gs*** is the number of valid solutions per group of houses, ***g*** is the number of groups of houses, ***c*** is the number of genetic hyperparameter configurations, and ***r*** is the number of runs.

For SHP, given 935 houses in the dataset, a total of 935 SHP PF visualisations were created, by the jMetalPy and the bespoke Python code, giving a total of 1,870 SHP PF visualisations.

For GHP, given 31 groups of houses in the dataset, 64 MOEA genetic hyperparameter configurations and 2 runs executed, a total of 3,840 GHP PF visualisations were created, by the jMetalPy and the bespoke Python code, giving a total of 7,680 GHP PF visualisations. These numbers can be increased by increasing the *number of runs* or the *number of genetic hyperparameter configurations*.

The creation of other types of visualisations follow the same principles, but the overall numbers vary according to the values in the variables in the equation.

7.1.6 Intervention Counts Visualisations

For each PF, visualisations were prepared of the *counts of interventions* for the points of interest which are the ***extrema*** on the PF (i.e.; the ***minima***, ***knee-point*** and ***maxima)***, using bespoke Python code developed in conjunction with inbuilt Matplotlib functionality.

A sample of each of the resultant *Intervention Counts* visualisations are shown below;

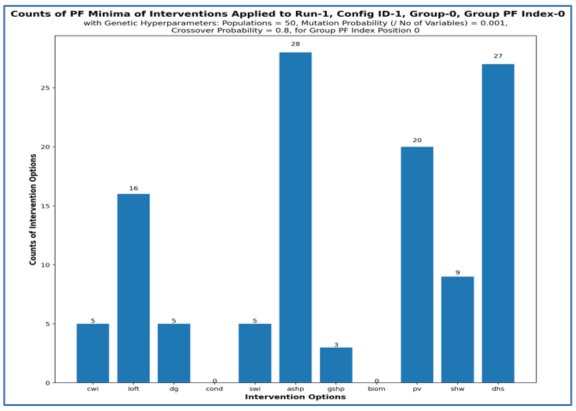


Figure 30. Intervention Count *Minima Values* for GHP PF

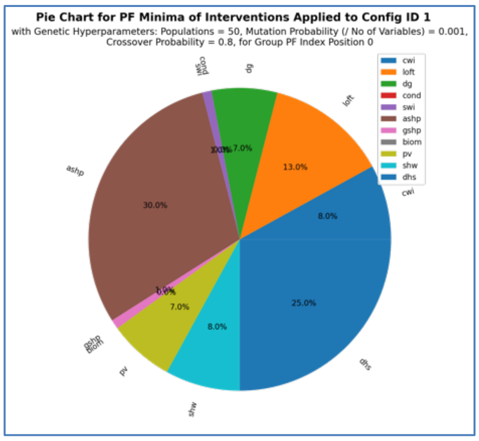


Figure 31. Intervention Count *Minima* % Pie Chart for GHP PF

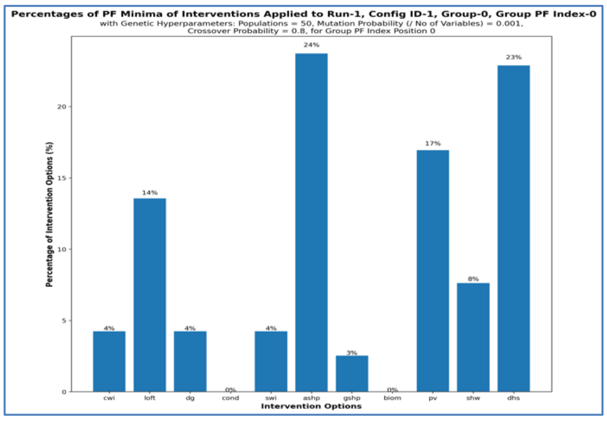


Figure 32. Intervention Count % for GHP PF

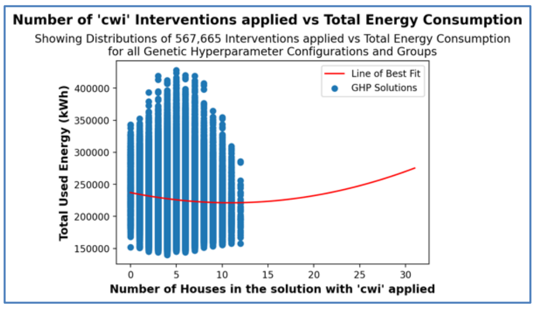


Figure 33. *'cwi'* Total Interventions applied vs Total Energy Consumption

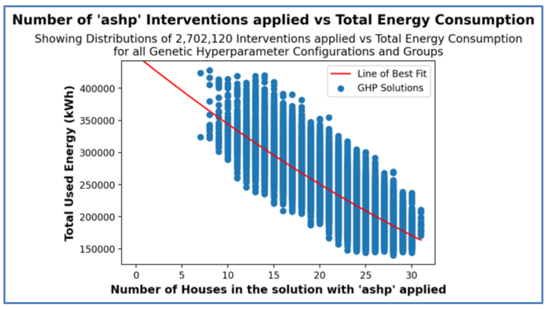


Figure 34. *'ashp'* Total Interventions applied vs Total Energy Consumption

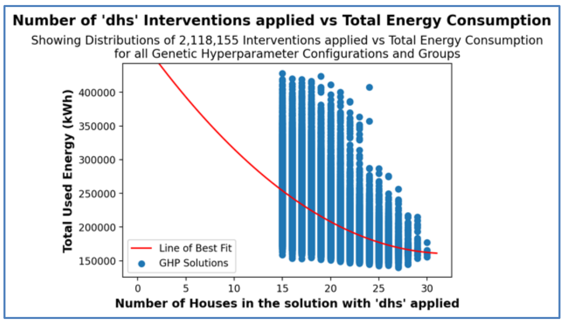


Figure 35. *'dhs'* Total Interventions applied vs Total Energy Consumption

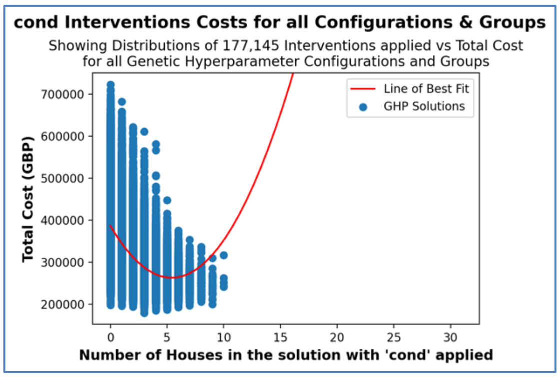


Figure 36. *'cond'* Total Interventions applied vs Total Cost

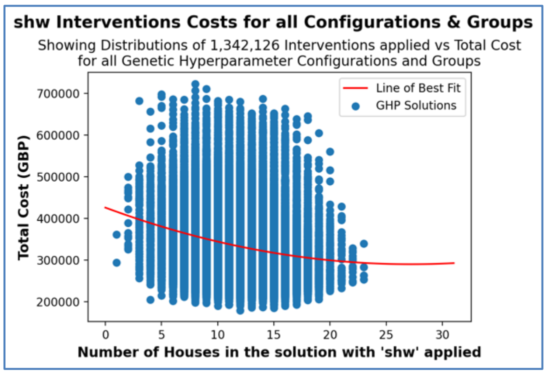


Figure 37. *'shw'* Total Interventions applied vs Total Cost

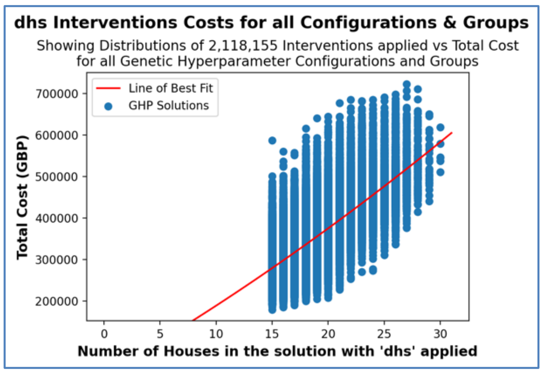


Figure 38. *'dhs'* Total Interventions applied vs Total Cost

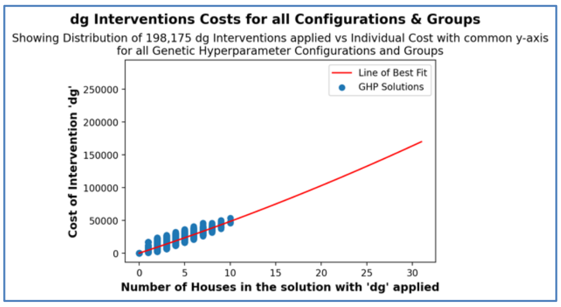


Figure 39. *'dg'* Total Interventions applied vs Total Cost – Common y-axis

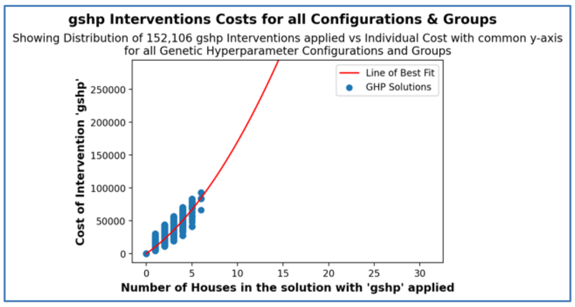


Figure 40. *'gshp'* **Total Interventions applied vs Total Cost – Common y-axis**

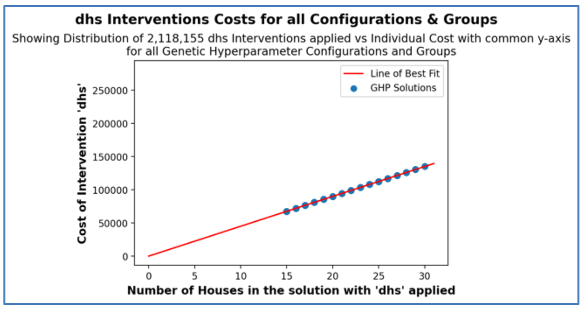


Figure 41. *'dhs'* **Total Interventions applied vs Total Cost – Common y-axis**

Ghani [69] describes the line of best fit (*used in the visualisations above*) as the statistical measure was used to show the trends in the data i.e.; trends that the naked human eye cannot easily see for the millions of decision-variables in this GHP results data.

For GHP, given 30 groups of houses in the dataset, 4 points of interest, 64 MOEA genetic hyperparameter configurations and 2 runs executed, a total of 11,520 GHP 'intervention counts' visualisations were created. These numbers can be increased by increasing the *number of runs* or the *number of genetic hyperparameter configurations*.

7.1.7 Intervention Count Comparisons Visualisations

For each PF, visualisations were prepared of the *comparison of intervention counts* for the points of interest which are the ***extrema*** on the PF (i.e.; the ***minima***, ***knee-point*** and ***maxima)***, using bespoke Python code developed in conjunction with inbuilt Matplotlib functionality.

A sample of each of the resultant *Intervention Count Comparisons* visualisations are shown below;

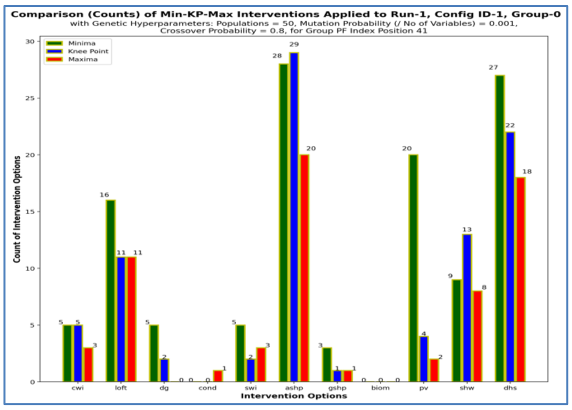


Figure 42. ***Extrema*** **Intervention Count** ***Values*** **Comparisons**

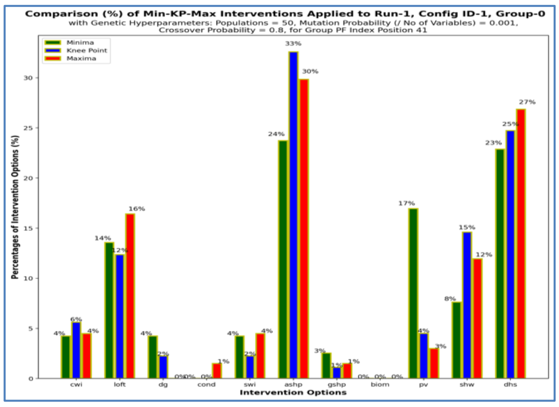


Figure 43. *Extrema* **Intervention Count % Comparisons**

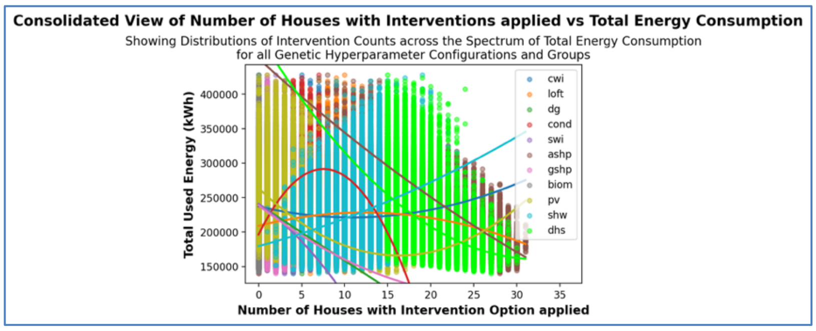


Figure 44. **Number of Houses in Groups with Interventions applied vs Total Energy**

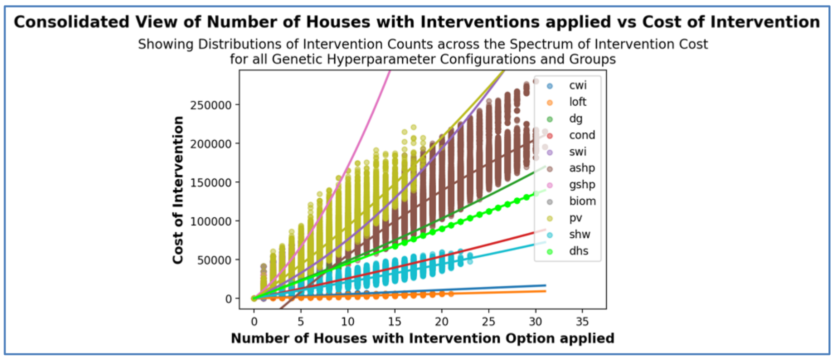


Figure 45. **Number of Houses in Groups with Interventions applied vs Total Costs**

Ghani [69] describes the line of best fit (*used in the visualisations above*) as the statistical measure was used to show the trends in the data i.e.; trends that the naked human eye cannot easily see for the millions of decision-variables in this GHP results data.

7.1.8 Aggregated Intervention Count Comparisons Visualisations

For each PF, visualisations were prepared of the *comparison of aggregated intervention counts* for the points of interest which are the ***extrema*** on the PF (i.e.; the ***minima***, ***knee-point*** and ***maxima)***, using bespoke Python code developed in conjunction with inbuilt Matplotlib functionality.

A sample of the resultant *Aggregated Intervention Count Comparisons* visualisations are shown below;

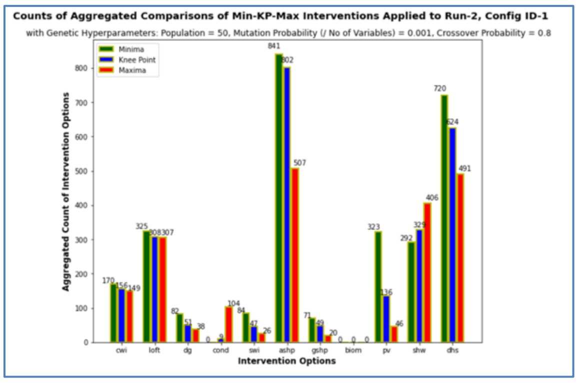


Figure 46. *Extrema* Aggregated Intervention Count *Values* Comparisons

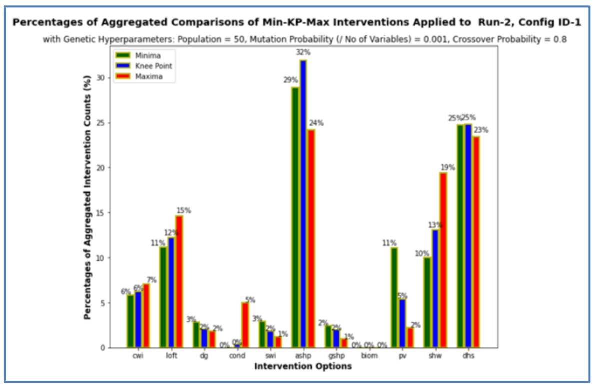


Figure 47. *Extrema* Aggregated Intervention Count *%* Comparisons

7.1.9 Genetic Hyperparameter Comparison Visualisations

Visualisations were prepared of the *comparison between genetic hyperparameter configurations results* for the points of interest which are the ***extrema*** on the PF (i.e.; the ***minima***, ***knee-point*** and ***maxima)***, using bespoke Python code developed in conjunction with inbuilt Matplotlib functionality.

A sample of resultant *Genetic Hyperparameter Configurations Comparisons* visualisations are shown below;

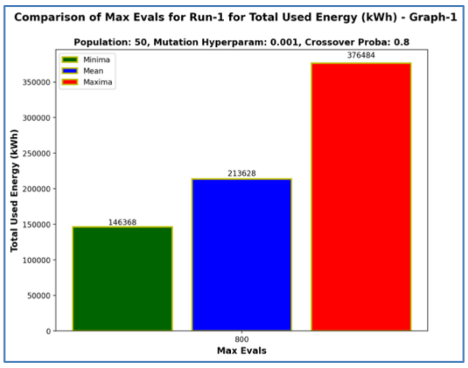


Figure 48. GH *Max Evaluations* vs Energy Comparisons

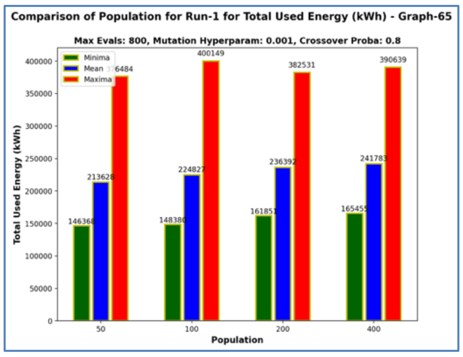


Figure 49. **GH *Population* vs Energy Comparisons**

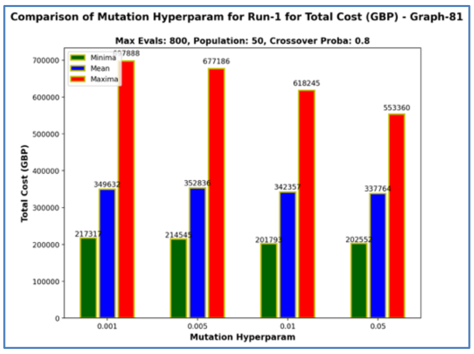


Figure 50. **GH *Mutation* vs Cost Comparisons**

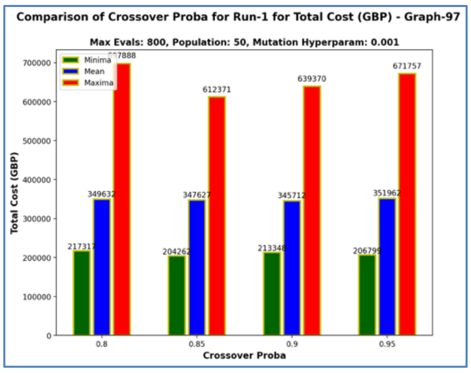


Figure 51. **GH *Crossover* vs Cost Comparisons**

7.1.10 Hypervolume Results Visualisations

For each GHP PF, a new GHP PF visualisation was created with HV as a *shaded area above the curve of the GHP PF line* relative to the RP, denoting MOEA efficacy. This was done with bespoke Python code with inbuilt Matplotlib functionality and HV scores as data inputs. A sample of the resultant GHP PF visualisations with HV are shown below;

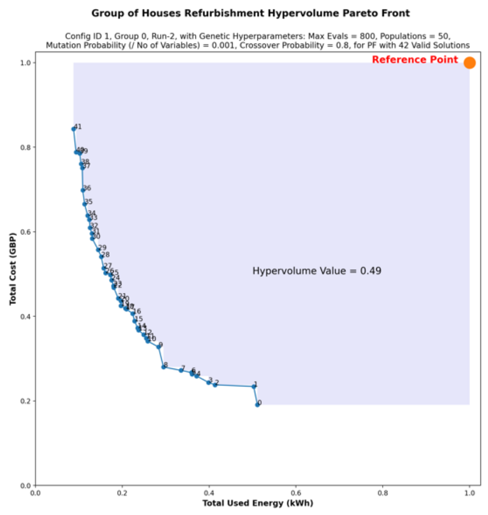


Figure 52. PF with Normalised HV

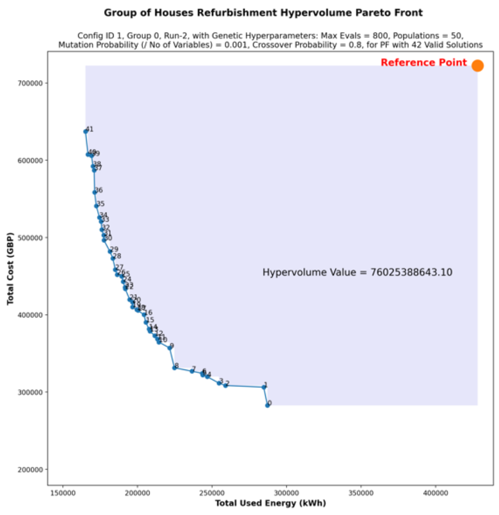
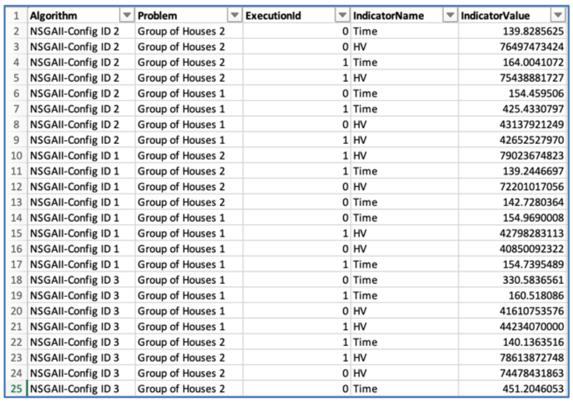


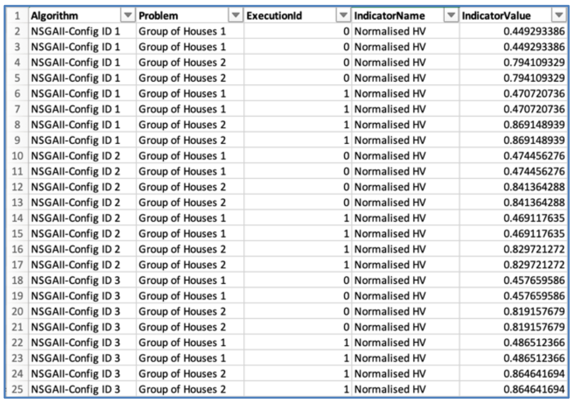
Figure 53. PF with Non-normalised HV

In Figure 54 (*non-normalised*) and Figure 55 (*normalised*) below, are the tabular HV results subset from which the GHP PF's above were created.



1	Algorithm	Problem	ExecutionId	IndicatorName	IndicatorValue
2	NSGAII-Config ID 2	Group of Houses 2	0	Time	139.8285625
3	NSGAII-Config ID 2	Group of Houses 2	0	HV	76497473424
4	NSGAII-Config ID 2	Group of Houses 2	1	Time	164.0041072
5	NSGAII-Config ID 2	Group of Houses 2	1	HV	75438881727
6	NSGAII-Config ID 2	Group of Houses 1	0	Time	154.459506
7	NSGAII-Config ID 2	Group of Houses 1	1	Time	425.4330797
8	NSGAII-Config ID 2	Group of Houses 1	0	HV	43137921249
9	NSGAII-Config ID 2	Group of Houses 1	1	HV	42652527970
10	NSGAII-Config ID 1	Group of Houses 2	1	HV	79023674823
11	NSGAII-Config ID 1	Group of Houses 2	1	Time	139.2446697
12	NSGAII-Config ID 1	Group of Houses 2	0	HV	72201017056
13	NSGAII-Config ID 1	Group of Houses 2	0	Time	142.7280364
14	NSGAII-Config ID 1	Group of Houses 1	0	Time	154.9690008
15	NSGAII-Config ID 1	Group of Houses 1	1	HV	42798283113
16	NSGAII-Config ID 1	Group of Houses 1	0	HV	40850092322
17	NSGAII-Config ID 1	Group of Houses 1	1	Time	154.7395489
18	NSGAII-Config ID 3	Group of Houses 1	0	Time	330.5836561
19	NSGAII-Config ID 3	Group of Houses 1	1	Time	160.518086
20	NSGAII-Config ID 3	Group of Houses 1	0	HV	41610753576
21	NSGAII-Config ID 3	Group of Houses 1	1	HV	44234070000
22	NSGAII-Config ID 3	Group of Houses 2	1	Time	140.1363516
23	NSGAII-Config ID 3	Group of Houses 2	1	HV	78613872748
24	NSGAII-Config ID 3	Group of Houses 2	0	HV	74478431863
25	NSGAII-Config ID 3	Group of Houses 2	0	Time	451.2046053

Figure 54. QI (HV) Test Results – Non-Normalised



1	Algorithm	Problem	ExecutionId	IndicatorName	IndicatorValue
2	NSGAII-Config ID 1	Group of Houses 1	0	Normalised HV	0.449293386
3	NSGAII-Config ID 1	Group of Houses 1	0	Normalised HV	0.449293386
4	NSGAII-Config ID 1	Group of Houses 2	0	Normalised HV	0.794109329
5	NSGAII-Config ID 1	Group of Houses 2	0	Normalised HV	0.794109329
6	NSGAII-Config ID 1	Group of Houses 1	1	Normalised HV	0.470720736
7	NSGAII-Config ID 1	Group of Houses 1	1	Normalised HV	0.470720736
8	NSGAII-Config ID 1	Group of Houses 2	1	Normalised HV	0.869148939
9	NSGAII-Config ID 1	Group of Houses 2	1	Normalised HV	0.869148939
10	NSGAII-Config ID 2	Group of Houses 1	0	Normalised HV	0.474456276
11	NSGAII-Config ID 2	Group of Houses 1	0	Normalised HV	0.474456276
12	NSGAII-Config ID 2	Group of Houses 2	0	Normalised HV	0.841364288
13	NSGAII-Config ID 2	Group of Houses 2	0	Normalised HV	0.841364288
14	NSGAII-Config ID 2	Group of Houses 1	1	Normalised HV	0.469117635
15	NSGAII-Config ID 2	Group of Houses 1	1	Normalised HV	0.469117635
16	NSGAII-Config ID 2	Group of Houses 2	1	Normalised HV	0.829721272
17	NSGAII-Config ID 2	Group of Houses 2	1	Normalised HV	0.829721272
18	NSGAII-Config ID 3	Group of Houses 1	0	Normalised HV	0.457659586
19	NSGAII-Config ID 3	Group of Houses 1	0	Normalised HV	0.457659586
20	NSGAII-Config ID 3	Group of Houses 2	0	Normalised HV	0.819157679
21	NSGAII-Config ID 3	Group of Houses 2	0	Normalised HV	0.819157679
22	NSGAII-Config ID 3	Group of Houses 1	1	Normalised HV	0.486512366
23	NSGAII-Config ID 3	Group of Houses 1	1	Normalised HV	0.486512366
24	NSGAII-Config ID 3	Group of Houses 2	1	Normalised HV	0.864641694
25	NSGAII-Config ID 3	Group of Houses 2	1	Normalised HV	0.864641694

Figure 55. QI (HV) Test Results – Normalised

7.1.11 Consolidated Comparisons View of PF's

Four consolidated views of PF comparisons by *genetic hyperparameter configuration* vs by *groups of houses* were created, as shown below;

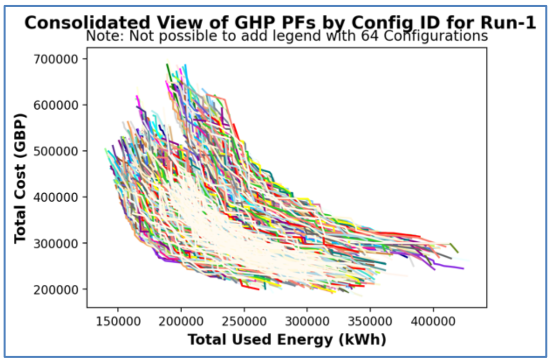


Figure 56. Run-1 – Consolidated View of PF's by Genetic Hyperparameter Configuration

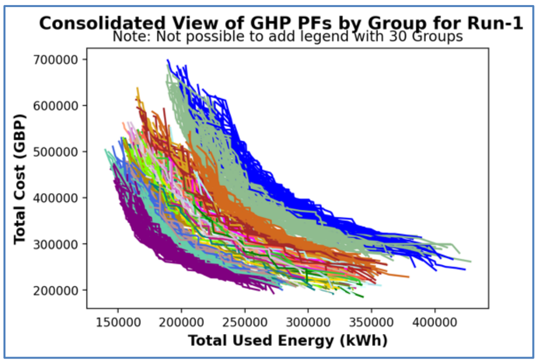


Figure 57. Run-1 – Consolidated View of PF's by Groups of Houses

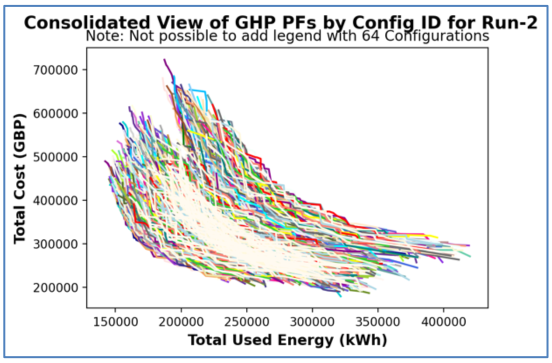


Figure 58. Run-2 – Consolidated View of PF's by Genetic Hyperparameter Configuration

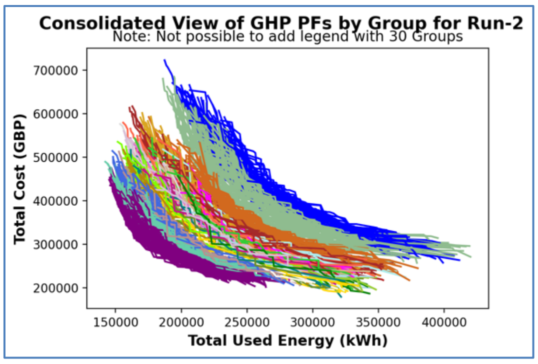


Figure 59. Run-2 – Consolidated View of PF's by Groups of Houses

Whilst the visualisations from both runs consistently show no discernible patterns arising from the *genetic hyperparameter configurations*, they do show clear discernible patterns from the *group of houses view*, that clearly show colour concentration density and a drift for the *group of houses view* that suggests that the *composition of the constituent elements* in GHP solutions are of more relevance to the GHP PF and by extension, to MOEA efficacy, than *genetic hyperparameter configurations*, for which there is no discernible pattern in either concentration or the drift.

This determination arises because both runs were performed by; simulating identical *non-uniform constant probabilities* for each of the genetic hyperparameter configurations, under an identical and modest *constant maximum evaluations* of 800 for each GHP NSGA-II function call, all as described in Section 5.3.

7.1.12 Search Space Size vs Runtime Visualisations

Two search space size vs runtime visualisations were created, as shown below;

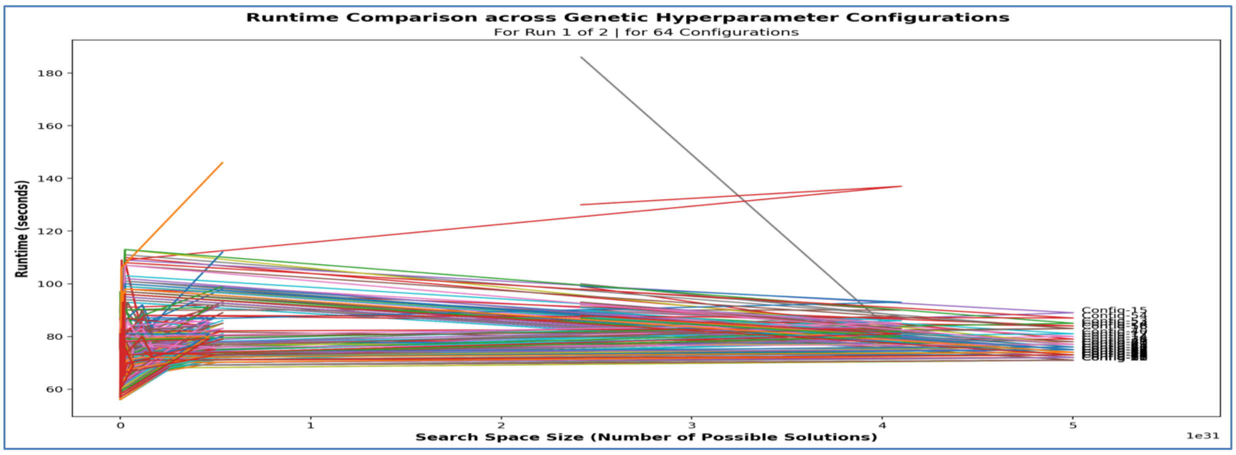


Figure 60. Run-1 – Runtime by Genetic Hyperparameter Configuration

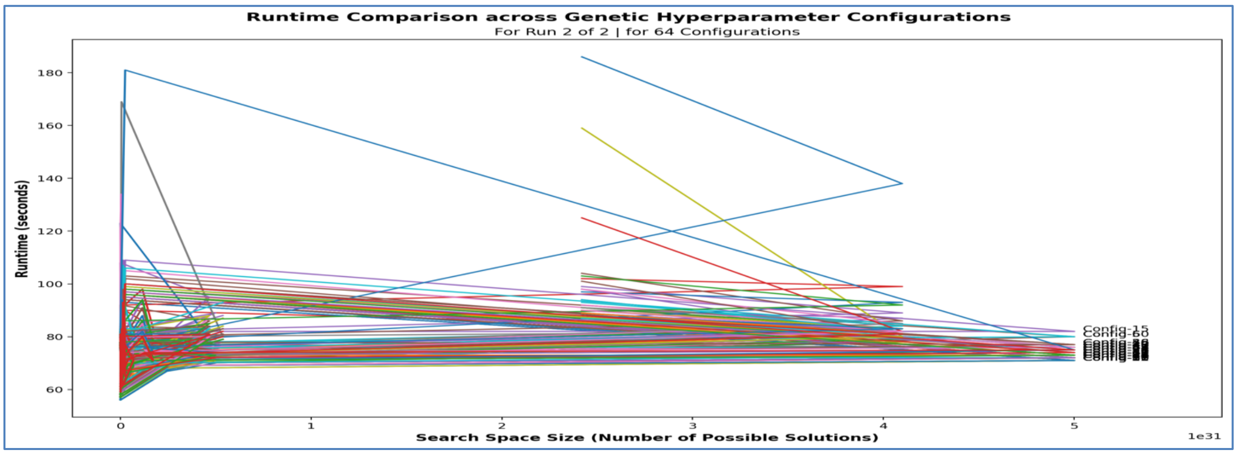


Figure 61. Run-2 – Runtime by Genetic Hyperparameter Configuration

Both visualisations show the self-learning properties of the NSGA-II function calls, across both runs, as their runtime efficacy (*reducing from approx.. the 60 to 110 seconds range per NSGA-II function call, to approx. 70 to 90 seconds range*) shows near linear runtime improvement as the search space size increases from *0 to 5.00e+31*. In effect, counter intuitively, the runtime reduced as search space size increased, and the pattern is discernible for most of the genetic hyperparameter configurations.

Due to time limitations, this project did not identify the genetic hyperparameter configurations that broke this trend. Nonetheless, in *future work* in this area should study the constituent intervention options in the GHP solutions in those genetic hyperparameter configurations, to find out whether the mix of interventions are responsible for this small variation to the otherwise dominant pattern of NSGA-II self-learning, as described in Section 5.5.2.

7.2 Discussions

The project started out as a software development project, limiting the scope of the literature review. In hindsight, it should have started as a research project, which would have resulted in less rework in the software development, albeit conscious inclusion was at the centre of its development practises.

7.2.1 Accomplishments vs Expectations

There was significant computation resource required to run the jMetalPy code, including extremely long runtime in excess of 12 hours for SHP, 5 days for GHP and 4 days for HV respectively. For LSGO research therefore, much more powerful computers are required, at least five times more powerful than the iMac used. Notwithstanding, the project was able to develop the software it set out to do and achieve all of the project goals.

Due to the selection of only already fine-tuned genetic hyperparameter values [4], [49], [25], there was no discernible impact identified from the 64 genetic hyperparameter configurations used. In effect, their potential impact was negated because they were already setup for peak performance.

With the exception of '*biom*', all the other variables had at least <0.05 significance, as can be seen in the visualisations showing (a) *Interventions Applied vs Total Energy Consumption* and (b) *Interventions Applied vs Total Cost*, in Section 7.1.6.

The line of best fit polynomial for '*dhs*' shows viability for GHP solutions with >8 houses in a group having '*dhs*' thermal efficiency interventions applied. However, this optimisation potential has been throttled by the '*>50% i.e.; 15 of houses in a group for a viable solution*' enforced constraint for '*dhs*' to be a valid intervention in a GHP solution. As a result, fewer '*dhs*' interventions have been being applied than would otherwise have been without the aforementioned enforced constraint. An initial assumption was that '*dhs*' could only be viable when applied to at least 50% of the houses in GHP, but the computational simulation shows that '*dhs*' is actually viable with a minimum of 8 out of 31 houses in a group, which is 25.8% of the houses in a group.

Enforced constraints unduly influenced the distributions of interventions applied in *non-dominated solutions*, particularly at the GHP level. For example, due to a combination of factors, there were no '*zero solutions*' on the PF's, because every house had at least one intervention option applied. The reasons for this are varied, but include;

1. Initial solution used ***random values*** rather than ***zero's***, plus original dataset was seeded, and 800 generations threshold per NSGA-II function call was sufficiently large to find better solutions that dominate the zero solutions.
2. The ***genetic hyperparameter configurations*** used were finely tuned from previous research.
3. The specified constraints implemented meant that every house would have at least one thermal efficiency intervention. These constraints include;
    a. Constraints in the data e.g.; ***negative cost values*** for renewables and '*dhs*' meant that they were more optimal solutions to the objective functions than a '*no cost*' option?
    b. Algorithmic constraints e.g.; > ***15 houses in solution*** for '*dhs*' meant that they were more optimal solutions to the objective functions than a '*no cost*' solution?
4. The solutions presented in the input data were already optimised by the previous computational simulation performed by Brownlee et.al [3].

Whilst the above considerations helped to affirmatively address the RQ's on one hand, they took away the opportunity to study factors that may lead to no solutions being applied to SHP or GHP.

This projects approach to LSGO problem decomposition differs slightly from Brownlee et.al's [3] novel encoding approach, because of the inclusion of a GHP sub-problem intermediate tier, as discussed in the Literature Review in Section 3.3. In the process, this project further simplified computation and significantly reduced the minimum capacity of computation resources required to solve the LSGO global-problem.

As the first step to explainability, this project used the PEP 8 Python coding standards [70] as the first step to ensure code transparency and readability as a foundational basis for tackling the opaqueness inherent in black-box metaheuristics-based MOEA.

Whilst efficient algorithmic design led to efficient code, including efficient results data collation and saving for SHP, the same could not be said entirely for GHP, because of the need to iteratively save results after each separate GHP function call.

This also applied to HV, for which it was not possible to save the HV jobs into an external data structure which would have simplified computation. This is because of reliance on streaming using Pickle which did not work. By then it was too late to switch to JSON.

This paper highlights many notable observations. However, this project consciously chose to exclude deep dive analysis of the composition of intervention options in SHP and GHP solutions because due to time constraints, it is best left the next phase of this research i.e.; in future work.

8 Conclusions

Finally, whilst further research and investigation is clearly required and is indeed necessary, this paper provides optimism of the viability of developing MOEA software for LSGO problems, particularly of housing stock optimisation use cases.

8.1 Findings

The findings include the summary of achievements presented in Section 1.4, and summarised insights discussed in Section 7.1, Section 7.2, using causal inferences drawn from across all the prior chapters.

The findings this project demonstrates vs the ***research questions*** outlined in Section 1.3 were;

1. ***RQ-1*** – Yes, LSGO problems can be solved with MOEA developed in jMetalPy.
2. ***RQ-2*** – Yes, MOEA developed in jMetalPy can provide satisfactory efficacy measured by HV quality indicator and reliability as measured by >12 days uninterrupted runtime.
3. ***RQ-3*** – Yes, MOEA developed in jMetalPy, to incorporate satisfactory explainability through (a) causal inference, (b) readable code, (c) testing and (d) externally saving results for subsequent post-execution review and analysis.
4. ***RQ-4*** – The main factors that influence efficacy and reliability of MOEA are; (a) computational resources, (b) problem decomposition via separability in algorithmic design, (c) efficient systematic algorithmic implementation that optimises computational resources, especially memory release, (d) quality/composition of solutions, (e) efficiency in genetic hyperparameter configuration, and (f) incorporation of unit testing in the code, to verify outcomes.

The key findings in relation to factors affecting ***computational resource utilisation*** were;

5. ***Processor/CPU Utilisation*** – (a) Data dimensions, especially number of rows and number of target variables; plus (b) Algorithmic complexity exemplified by (i) efficient coding e.g.; using loops and branches, and efficient (ii) genetic operator values selection e.g.; *mutation* and *crossover*, and (iii) non-genetic operators, especially NSGA-II's '*max_evaluations*' threshold variable.
6. ***Memory/DRAM Utilisation*** – (a) Data dimensions, particularly number of rows and number of target variables; plus (b) Algorithmic complexity exemplified by (i) efficient coding e.g.; incorporating *memory release*, and (ii) reducing *size of data structures* and programs e.g.; via splitting Notebooks to mitigate against *memory saturation* as data structures grow in memory.
7. ***Memory Efficiency*** – RAM grab and release was fully implemented for SHP, but partially for GHP because QI *jobs* object could not be released into an alternate data structure using Pickle.
8. ***Time Efficiency*** – Implemented for SHP (*see above*) with a single write to CSV statement .

The key findings in relation to factors affecting ***MOEA efficacy*** were;

1. ***Non-Genetic Operators Parameters in NSGA-II*** – Non-genetic operator parameters such as '*max_evaluations*' and bespoke *constraints* applied through supporting abstract sub-classes e.g.; reducing costs for thermal efficiency interventions that use renewable energy; affect the *composition of intervention options in the non-dominated solutions*, which in turn affects MOEA efficacy.
2. ***Algorithmic Design*** – Data dimensions including number of rows and target variables, algorithmic complexity vs search space size that determines lower and upper bounds used in *non-dominated solutions*, efficient coding e.g.; using loops and branches, and effective use of genetic operator values particularly, *mutation* and *crossover*, can all affect MOEA efficacy.
3. ***Problem Decomposition*** – Adding a third tier to LSGO global-problem decomposition within a systematic approach proved that it is possible to affect MOEA efficacy global-problem decomposition into smaller meaningful and much more manageable pieces rather than attempt to deal with much larger and usually intractable monoliths.

8.2 Evaluation

Steeped in the causal inference ethos, the project methodology outlined in Figure 12 in Section 5.1 was systematically followed iteratively, akin to how a programmer debugs code.

Combining CI with traditional software engineering techniques, was particularly satisfying, because it makes use of earlier computer science learning with more recent data science learning. Whilst the baseline project goals stated in the original project proposal were achieved early on, the subsequent results analysis using causal inference was simultaneously much more challenging and satisfying.

Studying MOEA as applied to LSGO and developing viable MOEA software for the housing stock optimisation Use Case were the two most satisfying aspects of this project.

This is reflected achievements relative to project goals outlined in Section 1.4, the contextual perspectives provided in Chapter 2, the reflective discussions in Section 7.2, the findings outlined in Section 8.1. and the concluding lessons and remarks provided in the remainder of this Chapter 8.

From the set of results visualisations depicted in Figure 56, Figure 57, Figure 58 and Figure 59, all in Section 7.1.11, it is clear that the constituent elements of GHP solutions were the most significant factor in MOEA efficacy as represented by PF proximity to the nadir point, suggesting that the choice of intervention options had more to do with MOEA efficacy than differences in genetic hyperparameter configurations. The intervention options that had the least significance had the line of best fit trending negatively.

8.3 Future Work

Whilst this project successfully achieved all of its set out goals, the notable breakthrough is successfully using jMetalPy as the centrepiece for the MOEA software developed. This shall help bring this research area closer to other researchers in the Python programmer community. Given the significant computational resource utilisation, this project has identified potential further integration with pyRAPL to help research how to optimise computational resource utilisation and energy.

Nonetheless, these complimentary considerations and additional research work that are worthy of further investigation, as they shall help advance the research already undertaken. They are;

1. Implement the third tier i.e.; LSGO global-problem as a new super-class in jMetalPy.
2. Satisfy MOEA objectives whilst minimising runtime e.g.; with more granular decomposition.
3. Incorporate practitioner considerations into the analysis, e.g.; societal benefits, budgetary considerations (*what do I get for my budget ?*), etc…
4. Incorporate all QI available in jMetalPy in future work e.g.; a re-run of this project with additional MOEA and all the aforementioned QI is desirable.
5. Improve memory deactivation at all stages of MOEA implementation, especially for long runtime, including results analysis, so as to optimise computational resources.
6. Avoid creating monolithic MOEA software for LSGO with long runtime, including for analysis of results,, and instead create smaller more optimal code blocks of Python programs, so as to operate runtime at optimum computation rather than at maxed computational capability.
7. Further investigation of the self-learning properties of MOEA as described in Section 5.5.2 and Sub-section 7.1.11, that resulted in less runtime to deal with larger search space sizes, which this project did not take to conclusion is warranted in future work. There is a suggestion that this may be due to the constituent intervention options in GHP solutions rather than genetic hyperparameter configurations. However, this should be studied further i.e.; why did some NSGA-II genetic hyperparameter configurations not exhibit self-learning with respect to runtime improvement vs search space size?

8. Raise JMetalPy tickets, to request;

    a. jMetalPy Matplotlib object be made extensible, so that new visualisations can be developed within jMetalPy, rather out with jMetalPy, as was done by this project.

    b. Pickle, JSON encoder and HMAC integration with jMetalPy's for QI Experiment.

8.4 Lessons Learned

Experience is a wonderful teacher, and helps to reflect on the many lessons learned;

1. It is a steeper learning curve to master all the moving parts that made up this project, particularly using multiple sets of genetic hyperparameter configurations. As a result, there were many missed opportunities that have now been earmarked for future work.
2. API's are not perfect. For example;

    a. A basic project requirement to create additional visualisations could not be satisfied by jMetalPy, because its Matplotlib figure object could not be extended.

    b. It was discovered very late that the Python serialisation library Pickle doesn't work with jMetalPy's *jobs* object, which would have made it possible to reduce the size of the most resource intensive Notebook created by this project. By the time of this realisation, it was too late to opt for the JSON encoder or even HMAC instead.

Further lessons are described in Sub-section 8.4.1, Sub-section 8.4.2 and Sub-section 8.4.3 below.

8.4.1 Personal Challenges

Whilst this project give a lot of personal satisfaction due to successfully achieving all the goals set, there were some personal challenges which are summarised below;

1. Mentally and emotionally challenging living and working in self-isolation as a vulnerable person at high risk to COVID.
2. Frustration of working in isolation without the natural collaboration with fellow postgraduate students and interaction with academic and teaching staff, due to COVID.
3. Without access to powerful university computers, lack of computational effort was a big challenge because of substantial runtime requirements, particularly for of;
    - Notebook-2 – 1.6 days runtime for 3 runs.
    - Notebook-3 – GHP only – 7.75 days runtime for 3 runs.
    - Notebook-3 – QI Experiment only – 6.45 days runtime for 3 runs.
4. Ankle reconstruction surgery on 6 August 2021 and the subsequent 6-weeks recovery whilst using crutches was not helped by getting an infection that necessitate 2-weeks of antibiotics treatment. Overall, this disrupted the momentum built up in this project.
5. TexStudio (LaTeX) setup developed a bug, which delayed the write-up. Eventually, text editor choice was switched to MS Word which is less optimal for scientific dissertation write up.

8.4.2 What Worked Well

A lot was covered in this research project, and the following are notable;

1. Data understanding and data preparation techniques from ML to identify anomalies in the data were used at the outset. Data cleansing was then performed to fix the anomalies found before computation of the LSGO problems. As a result, data validation checks were subsequently not be required for the remainder of the research project.

2. Created a high-level software design outline; and identified the data structures to use for data collation and results data storage, for further analysis outside Python and jMetalPy. This meant that subsequent refactoring was not required.
3. Results data collation, collection and extraction in tabular format was incorporated at key stages in the computation, so as to enable analysis of the results post computation.
4. From the results data collected, further statistical analysis of the results can be performed to further aid decision making.
5. Prepared pseudocode and README files to explain the sequences and computational steps in bespoke software developed.
6. Consciously focused on making sure that causal inference was at the centre of all the project work, and thus address the explainability goal, including by addition of metadata into all of the file outputs from the post-MOEA review and analysis of results, which can help other LSGO researchers and housing policy makers alike more easily drill-down into the results.

8.4.3 What Should Have Been Done Differently

A lot was covered in this research project, however there observations below are worth noting.

To better conceptualise the research work, I should have at the outset, done the following;

1. Resolved the issues I had with TexStudio, the LaTeX tool that I had intended to use. That would have enabled me to start the Dissertation write-up much earlier than I eventually did.
2. Started with the literature review rather than straight into coding, as this would have minimised the need for late re-runs using more appropriate genetic hyperparameter values.
3. Started the Dissertation write up much earlier, rather than putting it aside until experimentation was nearly complete. However, this may have been moot, but for having to undergo ankle reconstruction surgery in August 2021.
4. Used more than the single dataset. This would have better tested whether the quality of data has an impact on MOEA efficacy.
5. Used more than the single MOEA used i.e.; NSGA-II. This would have better tested the effect of metaheuristics on MOEA efficacy.
6. Attended some seminars/workshops on (a) LSGO, and; (b) developing with jMetal/jMetalPy.
7. Considered the need for QI much earlier, to assess MOEA efficacy and robustness of their performance. The delay resulted in settling for only HV in the end.
8. Used J*SON* rather than *Pickle* to save the QI jobs and in the process, split Notebook-3 into two with (a) the first part running GHP and (b) the second part running the QI Experiment. Pickle failed to work for the QI jobs and so Notebook-3 ended up with very long runtime.

In order not to lose track of my thoughts on new thoughts from literature as I was coding, I should have at the outset, done the following;

1. Incorporated academic theory in my Lab Book more rather focus I had solving problems with Python coding.
2. Created flowcharts and with control flows and process flows to visually explain the various sequences in the bespoke Python code developed, particularly in results analysis.

In order to get a better understanding of impacts on computational resource utilisation, I should have at the outset, done the following;

1. Requested access to a more powerful university computer than I had access to, perhaps multiple 12-core processors running at >5.0GHz processors with >128GB RAM. I omitted to do this because I mistakenly thought using Google Colab Pro would be sufficient. However, the maximum runtime for all tiers of Google Colab is 11 hours with their reasonably powerful GPUs. So, I was actually better off with my own iMac at home!
2. Utilised pyRAPL toolkit to measure computation resource utilisation e.g.; runtime, memory and processor time; for each GHP NSGA-II run, so as to better understand the relationship between Search Space and genetic hyperparameter combinations.
3. Incorporated computational resource utilisation results data collation, collection and extraction in tabular format, after computation, to facilitate subsequent analysis of the results.
4. Incorporated memory deactivation in all aspects of MOEA software for LSGO with long runtime, including for analysis of results, so as to make better and more effective use of computational resources.
5. Created a Staging Area for Python scripts, so that I could get others to run the scripts and feedback on performance when run on computers with different configurations.

8.5 Summary

Notwithstanding the challenges, I was able to affirmatively answer all the research questions as described in the findings in Section 8.1.

Whenever the project touched upon something new, it seemed that there was still more to find. As an answer was found, so was a new question.

There is still much to discover in this area of research.

Bibliography

[1] K. Deb, A. Pratap, S. Agarwal and T. A. M. T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II.,” *IEEE transactions on evolutionary computation,* vol. 6, no. 2, pp. 182-197, 7 August 2002.

[2] I. Costa-Carrapiço, R. Raslan and J. N. González, “A systematic review of genetic algorithm-based multi-objective optimisation for building retrofitting strategies towards energy efficiency,” *Energy and Buildings,* vol. 210, p. 109690, 1 March 2020.

[3] A. E. Brownlee, J. A. Wright, M. He, T. Lee and P. McMenemy, “A novel encoding for separable large-scale multi-objective problems and its application to the optimisation of housing stock improvements,” *Applied Soft Computing,* vol. 96, p. 106650, November 2020.

[4] Z. Drezner and T. D. Drezner, “Biologically inspired parent selection in genetic algorithms,” *Annals of Operations Research,* vol. 287, no. 1, pp. 161-183, 22 July 2020.

[5] Department for Business, Energy & Industrial Strategy, “Guidance - Heat Networks,” UK Government, 28 September 2021. [Online]. Available: https://www.gov.uk/guidance/heat-networks-overview. [Accessed 30 September 2021].

[6] D. Soni, “Introduction to Evolutionary Algorithms - Optimization by natural selection,” *Towards Data Science,* 18 Feb 2018.

[7] Merriam-Webster, “Optimization,” Merriam-Webster, September 2021. [Online]. Available: https://www.merriam-webster.com/dictionary/optimization. [Accessed 24 September 2021].

[8] S. Rostami, F. Neri and K. Gyaurski, “On algorithmic descriptions and software implementations for multi-objective optimisation: A comparative study.,” *SN Computer Science,* vol. 1, no. 5, pp. 1-23, September 2020.

[9] S. Michailos, D. Parker and C. Webb, “Design, sustainability analysis and multiobjective optimisation of ethanol production via syngas fermentation,” *Waste and Biomass Valorization,* vol. 10, no. 4, pp. 865-876, April 2019.

[10] A. E. Eiben and S. K. Smit, “Parameter tuning for configuring and analyzing evolutionary algorithms,” *Swarm and Evolutionary Computing,* vol. 1, no. 1, pp. 19-31, March 2011.

[11] A. E. Eiben and J. E. Smith, Introduction to evolutionary computing, vol. 53, G. Rozenberg, T. Bäck, A. Eiben, J. Kok and H. Spaink, Eds., Berlin: Springer, 2003, p. 18.

[12] A. Gad, “Introduction to Optimization with Genetic Algorithm,” https://www.linkedin.com/in/ahmedfgad/, 3 July 2018. [Online]. Available: https://towardsdatascience.com/introduction-to-optimization-with-genetic-algorithm-2f5001d9964b. [Accessed March 2021].

[13] K.-H. Chang, “Chapter 5 - Multiobjective Optimization and Advanced Topics,” in *Design Theory and Methods using CAD/CAE*, Amsterdam, Elsevier, ScienceDirect, 2015.

[14] J. A. Manson, T. W. Chamberlain and R. A. Bourne, “MVMOO: Mixed variable multi-objective optimisation,” *Journal of Global Optimization,* vol. 80, pp. 865-886, 9 July 2021.

[15] X. Ma, F. Liu, Y. Qi, X. Wang, L. Li, L. Jiao, M. Yin and M. Gong, “A multiobjective evolutionary algorithm based on decision variable analyses for multiobjective optimization problems with large-scale variables,” *IEEE Transactions on Evolutionary Computation,* vol. 20, no. 2, pp. 275-298, 13 July 2015.

[16] T.-L. Yu, D. E. Goldberg, K. Sastry, C. F. Lima and M. Pelikan, “Dependency structure matrix, genetic algorithms, and effective recombination,” *Evolutionary computation,* vol. 17, no. 4, pp. 595-626, December 2009.

[17] F. Bre and V. D. Fachinotti, “A computational multi-objective optimization method to improve energy efficiency and thermal comfort in dwellings,” *Energy and Buildings,* vol. 154, pp. 283-294, 2 August 2017.

[18] S. Khoshnevisan, W. Gong, L. Wang and C. H. Juang, “Robust design in geotechnical engineering–an update,” *Georisk: Assessment and Management of Risk for Engineered Systems and Geohazards,* vol. 8, no. 4, pp. 217-234, 2014.

[19] M. He, A. Brownlee, J. A. Wright and S. Taylor, “Multi-dwelling Refurbishment Optimization: Problem Decomposition, Solution, and Trade-off Analysis,” in *4th International Conference of the International Building Performance Simulation Association (BS2015)*, Hyderabad, 2015.

[20] M. He, A. Brownlee, T. Lee, J. Wright and S. Taylor, “Multi-objective optimization for a large scale retrofit program for the housing stock in the North East of England,” *Energy Procedia,* vol. 78, pp. 854-859, 1 November 2015.

[21] P. J. Wright and D. A. Brownlee, “Introduction to Building Stock Optimization and Sequential Pareto Optimization,” Loughborough University, Loughborough, 2021.

[22] S. Meyer-Nieberg and H.-G. Beyer, “Self-adaptation in evolutionary algorithms,” *Parameter setting in evolutionary algorithms,* pp. 47-75, 2007.

[23] H. Zille, H. Ishibuchi, S. Mostaghim and Y. Nojima, “A framework for large-scale multiobjective optimization based on problem transformation,” *Zille, H., Ishibuchi, H., Mostaghim, S., & Nojima, Y. (2017). A framework for large-scale multiobjective optimization based on problem transformation. IEEE Transactions on Evolutionary Computation,* vol. 22, no. 2, pp. 260-275, 16 May 2018.

[24] D. M. Cabrera, “Evolutionary algorithms for large-scale global optimisation: a snapshot, trends and challenges.,” *Progress in Artificial Intelligence,* vol. 5, no. 2, pp. 85-89, 1 May 2016.

[25] Z. Zhao, B. Liu, C. Zhang and H. Liu, “An improved adaptive NSGA-II with multi-population algorithm,” *Applied Intelligence,* vol. 49, no. 2, pp. 569-580, 6 September 2019.

[26] R. C. Purshouse, “On the evolutionary optimisation of many objectives,” University of Sheffield, Sheffield, 2003.

[27] G. Anadiotis, “DeepMind aims to marry deep learning and classic algorithms,” VentureBeat, 10 September 2021. [Online]. Available: https://venturebeat.com/2021/09/10/deepmind-aims-to-marry-deep-learning-and-classic-algorithms/. [Accessed September 2021].

[28] A. S. Azad, M. S. A. Rahaman, J. Watada, P. Vasant and J. A. G. Vintaned, “Optimization of the hydropower energy generation using meta-heuristic approaches: A review.,” *Energy Reports,* vol. 1, no. 6, pp. 2230-2248, 10 November 2020.

[29] Q. Zhang and H. Li, “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition,” *IEEE Transactions on Evolutionary Computation,* vol. 11, no. 6, pp. 712-731, December 2007.

[30] A. E. I. Brownlee, A. Wallace and D. Cairns, “Mining Markov Network Surrogates to Explain the Results of Metaheuristic Optimisation,” in *Proceedings of the SICSA eXplainable Artifical Intelligence Workshop 2021*, Aberdeen, 2021.

[31] C. Waibel, T. Wortmann, R. Evins and J. Carmeliet., “Building energy optimization: An extensive benchmark of global search algorithms,” *Energy and Buildings,* vol. 187, pp. 218-240, 15 March 2019.

[32] P. Westermann and R. Evins, “Surrogate modelling for sustainable building design–a review,” *Energy and Buildings,* vol. 198, pp. 170-186, 1 September 2019.

[33] K. Li, “Decomposition Multi-Objective Evolutionary Optimization: From State-of-the-Art to Future Opportunities,” *Neural and Evolutionary Computing,* vol. https://arxiv.org/abs/2108.09588, no. Preprint, 21 August 2021.

[34] A. Kulkarni and S. Puntambekar, “Efficacy of genetic algorithms for computationally intractable problems,” in *Modern Optimization Methods for Science, Engineering and Technology*, Vols. http://dx.doi.org/10.1088/978-0-7503-2404-5ch16, IOP Publishing, 2019.

[35] X. Li, “Decomposition and cooperative coevolution techniques for large scale global optimization,” in *In Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, New York, 2014.

[36] M. A. Potter and K. A. D. Jong, “A cooperative coevolutionary approach to function optimization,” in *In International Conference on Parallel Problem Solving from Nature*, Berlin, 1994.

[37] A. J. Nebro, “jMetalPy - Python version of the jMetal framework,” Massachusetts Institute of Technology (MIT), 12 June 2018. [Online]. Available: https://github.com/jMetal/jMetalPy. [Accessed March 2021].

[38] A. Benítez-Hidalgo, A. J. Nebro, J. García-Nieto, I. Oregi and J. D. Ser, “jMetalPy: A Python framework for multi-objective optimization with metaheuristics,” *Swarm and Evolutionary Computation,* vol. 51, p. 100598, 1 December 2019.

[39] A. Benítez-Hidalgo, “jMetalPy: Python version of the jMetal framework,” Massachusetts Institute of Technology (MIT), 2019. [Online]. Available: https://jmetal.github.io/jMetalPy/index.html. [Accessed 30 March 2021].

[40] ODSC - Open Data Science, “Why You Should be Using Jupyter Notebooks,” ODSC - Open Data Science, 15 July 2020. [Online]. Available: https://medium.com/@ODSC/why-you-should-be-using-jupyter-notebooks-ea2e568c59f2. [Accessed September 2021].

[41] Luke, Sean;, “Essentials of Metaheuristics (Lecture Notes),” Department of Computer Science, 2015. [Online]. Available: http://yalma.fime.uanl.mx/~roger/work/teaching/class_tso/docs/3-Metaheuristics/2015-Luke-Essentials%20of%20Metaheuristics.pdf. [Accessed March 2021].

[42] H. J. Muller, “The Darwinian and Modern Conceptions of Natural Selection.,” *Natural Selection and Adaptation,* vol. 93, no. 6, pp. 459-470, 29 December 1949.

[43] N. A. Zolpakar, M. F. Yasak and S. Pathak, “A review: use of evolutionary algorithm for optimisation of machining parameters,” *The International Journal of Advanced Manufacturing Technology volume ,* vol. 115, pp. 31-47, 4 May 2021.

[44] S. Manoharan, “Population based meta heuristics algorithm for performance improvement of feed forward neural network,” *Journal of soft computing paradigm (JSCP),* vol. 2, no. 1, pp. 36-46, March 2020.

[45] X. Hao, J. Liu and Z. Wang, “An improved global replacement strategy for MOEA/D on many-objective knapsack problems,” in *13th IEEE Conference on Automation Science and Engineering (CASE)*, Xi'an, China, 2017.

[46] B. A. Conway, “A survey of methods available for the numerical optimization of continuous dynamic systems,” *Journal of Optimization Theory and Applications,* vol. 152, no. 2, pp. 271-306, 1 February 2012.

[47] X.-S. Yang, “Genetic Algorithms (Chapter 5),” in *Nature Inspired Optimization Algorithms*, Amsterdam, Elsevier, ScienceDirect, 2014.

[48] P. Wu, Q. Yang, W. Chen, B. Mao and H. Yu, “An Improved Genetic-Shuffled Frog-Leaping Algorithm for Permutation Flowshop Scheduling,” *Complexity,* vol. 2020, no. Article ID 3450180, p. https://doi.org/10.1155/2020/3450180, 28 November 2020.

[49] A. Malik, “A study of genetic algorithm and crossover techniques,” *International Journal of Computer Science and Mobile Computing,* vol. 8, no. 3, pp. 335-344, March 2019.

[50] S. A. Stanhope and J. M. Daida, “Optimal mutation and crossover rates for a genetic algorithm operating in a dynamic environment,” in *International Conference on Evolutionary Programming VII*, Berlin, 1998.

[51] Q. Wang, L. Wang, W. Huang, Z. Wang, S. Liu and D. A. Savic, “Parameterization of NSGA-II for the optimal design of water distribution systems.,” *Water,* vol. 11, no. 5, p. 971, May 2019.

[52] J. Idol, “What is Crossover Probability & Mutation Probability in Genetic Algorithm or Genetic Programming?,” 27 March 2021. [Online]. Available: https://stackoverflow.com/questions/2877895/what-is-crossover-probability-mutation-probability-in-genetic-algorithm-or-gen. [Accessed September 2021].

[53] A. Maghawry, R. Hodhod, Y. Omar and M. Kholief, “An approach for optimizing multi-objective problems using hybrid genetic algorithms,” *Soft Computing,* vol. 25, no. 1, pp. 389-405, January 2021.

[54] J. Stalfort, “Hyperparameter tuning using Grid Search and Random Search: A Conceptual Guide,” Medium, 5 June 2019. [Online]. Available: https://medium.com/@jackstalfort/hyperparameter-tuning-using-grid-search-and-random-search-f8750a464b35. [Accessed September 2021].

[55] Tutorials Point, “Genetic Algorithms - Fitness Function,” Tutorials Point, 2021. [Online]. Available: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_fitness_function.htm. [Accessed September 2021].

[56] V. Mallawaarachchi, “How to define a Fitness Function in a Genetic Algorithm?,” Medium, 21 July 2017. [Online]. Available: https://towardsdatascience.com/how-to-define-a-fitness-function-in-a-genetic-algorithm-be572b9ea3b4. [Accessed September 2021].

[57] O. Tunali, A. T. Bayrak, V. Sanchez-Anguix and R. Aydoğan, “Multi-objective evolutionary product bundling: a case study,” in *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO)*, 2021.

[58] L. M. S. Russo and A. P. Francisco, “Extending quick hypervolume,” *Journal of Heuristics,* vol. 22, no. 3, pp. 245-271, 11 March 2016.

[59] Y. Cao, B. J. Smucker and T. J. Robinson, “On using the hypervolume indicator to compare Pareto fronts: Applications to multi-criteria optimal experimental design,” *Journal of Statistical Planning and Inference,* vol. 160, pp. 60-74, 1 May 2015.

[60] A. Auger, J. Bader, D. Brockhoff and E. Zitzler, “Theory of the hypervolume indicator: optimal μ-distributions and the choice of the reference point,” *In Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms,* vol. 10, pp. 87-102, 9 January 2009.

[61] İ. Demir, F. C. Ergın and B. Kıraz, “A new model for the multi-objective multiple allocation hub network design and routing problem,” *IEEE Access,* vol. 7, pp. 90678-90689, 8 July 2019.

[62] L. While, L. Bradstreet and L. Barone, “A fast way of calculating exact hypervolumes,” *IEEE Transactions on Evolutionary Computation,* vol. 16, no. 1, pp. 86-89, February 2012.

[63] J. Cohoon, J. Kairo and J. Lienig, “Evolutionary algorithms for the physical design of VLSI circuits,” in *Advances in Evolutionary Computing. Natural Computing Series*, Vols. pp 683-711, Berlin, Heidelberg, Springer, 2003, pp. 683-711.

[64] Q. Ye, Y. Sun, J. Zhang and J. Lv, “A Distributed Framework For EA-Based NAS,” *IEEE Transactions on Parallel and Distributed Systems,* vol. 32, no. 7, pp. 1753-1764, 23 December 2020.

[65] H. Maier, Z. Kapelan, J. Kasprzyk, J. Kollat, L. Matott, M. Cunha, G. Dandy, M. Gibbs, E. Keedwell, A. Marchi and A. Ostfeld, “Evolutionary algorithms and other metaheuristics in water resources: Current status, research challenges and future directions,” *Environmental Modelling & Software,* vol. 62, pp. 271-299, 1 December 2014.

[66] Wilfrid Laurier University Online, “How is Algorithm Design Applied?,” Wilfrid Laurier University Online, 12 February 2019. [Online]. Available: https://online.wlu.ca/news/2019/02/12/how-algorithm-design-applied. [Accessed September 2021].

[67] C. Chawla, “Trust in blockchains: Algorithmic and organizational,” *Journal of Business Venturing Insights,* vol. 14, p. e00203, 1 November 2020.

[68] M. Krasnov, “Abstract Classes and Meta Classes in Python,” everyday.codes, 21 July 2020. [Online]. Available: https://everyday.codes/python/abstract-classes-and-meta-classes-in-python/. [Accessed April 2021].

[69] A. Ghani, “Polynomial Regression in Tensorflow,” Analytics Vidhya, 16 December 2019. [Online]. Available: https://medium.com/analytics-vidhya/polynomial-regression-in-tensorflow-53f958cef2d2. [Accessed 24 September 2021].

[70] Python Community Contributors, “PEP 8 -- Style Guide for Python Code,” Python Software Foundation, 1 August 2013. [Online]. Available: https://www.python.org/dev/peps/pep-0008/. [Accessed March 2021].

Appendix 1 – Supplementary References

Supplementary materials including the project artefacts have been archived in university’s digital project repository. The details of these artefacts includes;

- 7 Jupyter notebook files.
- 1 input dataset in CSV format.
- 1 Single House Results – Double Length.csv.
- 1 Group of Houses Results.csv.
- 1 Aggregated Intervention Counts.csv.
- 1 QualityIndicatorSummary.csv.
- Several QI Experiments output files.

Appendix 2 – MOEA Software User guide

Pseudocode – Single House Sub-Problem (SHP)

Python libraries, toolkits, methods and pseudocode used for the computation in SHP are;

- **Python Libraries Used:** jMetalPy, Pandas, NumPy, Matplotlib, Sys, Time.
- **In-built Methods Used:** Specified in the Import Statements cell of the Notebooks.

Data Input: Updated CHM data sub-set with 935 houses.

Algorithm-1 Single House Sub-Problem (SHP) Algorithm

```
 1: INPUT updated CHM data sub-set → load into Pandas dataframe;
 2: explore and validate the data → data validation;
 3: calculate the number of houses in CHM dataset, by finding their start and end positions in the dataframe;
 4: SHP decomposition, by extending jMetalPy Binary Problem super class, abstract classes, and abstract methods;
 5: for i (number of Houses) ← 1 to 935 do
 6:         define SHP for single house
 7:         run NSGA-II algorithm with baseline metaheuristics configuration
 8:         collate results (SHP Pareto Front of non-dominated solutions)
 9:         create results visualisations (SHP Pareto Front of non-dominated solutions)
10:         save results visualisations to PNG (SHP Pareto Front of non-dominated solutions)
11:         calculate solution ranking (SHP PF Index)
12:         update list of overall solution ranking (Overall SHP PF Index) and overall solutions (Overall SHP PF Solutions)
13:         add populated SHP PF Index and SHP PF Solutions columns to the dataframe
14: end for
15: save results dataframe to CSV
```

Readme for Single House Sub-Problem (SHP)

Below is the Readme file for SHP (Notebook-2).

README	Readme File for Single House Problem (SHP), as part of M.Sc. AI Dissertation (ITNPBD5)
AUTHORS	Student ID: 2832057 (*M.Sc. in Artificial Intelligence*, 2020/21)
THANKS	To the authors and distributors of the Python Libraries, toolkits and methods outlined in the Pseudo-code in the preceding sub-section, as well as the developers and providers of the Anaconda Platform.
CHANGE LOG	Please refer to the Pseudocode in preceding sub-section and in-flight comments within Notebook-2.
INSTALL	Upload the Jupyter Notebook titled "2-Dissertation 2021 - 15 Oct - SHP.ipynb" to the root directory folder of your choice.
INPUT DATA	***CHM_RetrofitHouses_updated.csv***
TO RUN LO-CALLY	To run the full set of Jupyter Notebooks on a local PC; Choose a root directory on your local PC; Create sub-folders for organisational purposes, named as follows; • ***Data***, and copy the ***CHM_RetrofitHouses_updated.csv*** file to this location • ***PF/Single***, (2) ***PF/Group*** and (3) ***PF/Hypervolume*** • ***PF-jMetalPy/Single*** and (2) ***PF-jMetalPy/Group*** • ***Results CSVs*** • ***Intervention Counts***, (2) ***Comparisons*** and (3) ***GH Comparisons*** • ***Intervention Counts Aggregated***,(2) ***Comparisons Aggregated*** and (3) ***Comparison Means*** jMetalPy's 'Experiment' shall create the following outputs; • Sub-folder named (1) ***Experiment***, for intermediate outputs (*used later to calculate Hypervolume outputs*) • Output file named ***QualityIndicators.csv*** *in the root directory* Comment out the cells in the Notebook for running on Google Colab. Choose "***Restart & Run All***" under the Kernel menu in Jupyter Notebook.
TO RUN ON GOOGLE COLAB	To run the Jupyter Notebook on Google Colab; Choose a root directory on your Google Drive;

	Same as for *running on local PC*, above. Same as for *running on local PC*, above. Comment out the cells in the Notebook for running on a local PC. Choose "***Restart & Run All***" under the Kernel menu in Jupyter Notebook.
COPYING / LICENSE	For educational use only in the Division of Computing Science and Mathematics at the University of Stirling.
BUGS	No known bugs. Please refer to comments in the Notebook.
CONTRIBUTING	***Future Work*** has been added to a sub-section of the same name in the submitted Dissertation paper.

Pseudocode – Group of House Global-Problem (GHP)

Python libraries, toolkits, methods and pseudocode used for the computation in GHP are;

- **Python Libraries Used:** jMetalPy, Pandas, NumPy, Matplotlib, Sys, Time.
- **In-built Methods Used:** Specified in the Import Statements cell of the Notebooks.

Data Input: SHP optimisation results, in a file named; Single House Results – Double Length.csv.

Algorithm-2 Group of Houses Global-Problem (GHP) Algorithm

```
 1: INPUT Single House Sub-Problem (SHP) Results data → load into Pandas dataframe;
 2: calculate the number of groups in SHP Results data, by finding their start and end positions in the dataframe;
 3: GHP decomposition, by extending jMetalPy Integer Problem super class, abstract classes, and abstract methods;
 4: create dictionary of genetic hyperparameters → metaheuristics combinations;
 5: create results output CSV file → with target column names, including GHP PF Index and GHP PF Solutions
 6: for i (number of Runs) ← 1 to 2 do
 7:     for j (number of genetic hyperparameter configurations) ← 1 to 64 do
 8:         for k (number of Groups of Houses) ← 1 to 30 do
 9:             define GHP for a group of houses
10:             run NSGA-II algorithm for current Group of Houses, with currently specified
                    genetic hyperparameters configuration
11:             create a Quality Indicators job for NSGA-II preceding execution
12:             append a Quality Indicators job to a global list of Quality Indicators job to be performed later
13:             collate results (GHP Pareto Front of non-dominated solutions)
14:             create results visualisations (GHP Pareto Front of non-dominated solutions)
15:             save results visualisations to PNG (GHP Pareto Front of non-dominated solutions)
16:             save results tabular data to CSV (all columns of CSV)
17:         end for
18:     end for
19: end for
20: save results dataframe to CSV
```

Readme for Group of Houses Sub-Problem (GHP)

Below is the Readme file for SHP (Notebook-2).

README	Readme File for Single House Problem (SHP), as part of M.Sc. AI Dissertation (ITNPBD5)
AUTHORS	Student ID: 2832057 (*M.Sc. in Artificial Intelligence*, 2020/21)
THANKS	To the authors and distributors of the Python Libraries, toolkits and methods outlined in the Pseudocode in the preceding sub-section, as well as the developers and providers of the Anaconda Platform.
CHANGE LOG	Please refer to the Pseudocode in preceding sub-section and in-flight comments within Notebook-2.
INSTALL	Upload the Jupyter Notebook titled "2-Dissertation 2021 - 15 Oct - GHP.ipynb" to the root directory folder of your choice.
INPUT DATA	***CHM_RetrofitHouses_updated.csv***
TO RUN LOCALLY	To run the full set of Jupyter Notebooks on a local PC; Choose a root directory on your local PC; Create sub-folders for organisational purposes, named as follows; • ***Data***, and copy the ***CHM_RetrofitHouses_updated.csv*** file to this location • ***PF/Single***, (2) ***PF/Group*** and (3) ***PF/Hypervolume*** • ***PF-jMetalPy/Single*** and (2) ***PF-jMetalPy/Group***

	• ***Results CSVs*** • ***Intervention Counts***, (2) ***Comparisons*** and (3) ***GH Comparisons*** • ***Intervention Counts Aggregated***,(2) ***Comparisons Aggregated*** and (3) ***Comparison Means*** jMetalPy's 'Experiment' shall create the following outputs; • Sub-folder named (1) ***Experiment***, for intermediate outputs (*used later to calculate Hypervolume outputs*) • Output file named ***QualityIndicators.csv*** *in the root directory* Comment out the cells in the Notebook for running on Google Colab. Choose "***Restart & Run All***" under the Kernel menu in Jupyter Notebook.
TO RUN ON GOOGLE COLAB	To run the Jupyter Notebook on Google Colab; Choose a root directory on your Google Drive; Same as for *running on local PC*, above. Same as for *running on local PC*, above. Comment out the cells in the Notebook for running on a local PC. Choose "***Restart & Run All***" under the Kernel menu in Jupyter Notebook.
COPYING / LI-CENSE	For educational use only in the Division of Computing Science and Mathematics at the University of Stirling.
BUGS	No known bugs. Please refer to comments in the Notebook.
CONTRIBUTING	***Future Work*** has been added to a sub-section of the same name in the submitted Dissertation paper.

QI Experimentation Results Outputs from jMetalPy

The jMetalPy QI's experiment automatically by the jMetalPy ***Experiment*** class, creates the following output results;

1. **jMetalPy.log** file, with chronological order of the experiment's events and timings.
2. **QualityIndicators.csv** file, with results of the experiment, from which the GHP PF with the Reference Point and HV details is then plotted for visualisation purposes.
3. Nested sub-folder structure containing experiments results, named ***NSGAII-Config ID x***, where ***x*** = the genetic hyperparameters Configuration ID;
   a. Nested sub-folder structure (*Group of Houses **y***), where ***y*** = Group (ID).
      i. **VAR.*z*.tsv** – Variables in the job, per group, where ***z*** = Run number.
      ii. **FUN.*z*.tsv** – Functions in the job, per group, where ***z*** = Run number.
      iii. **TIME.*z*** – Several, where ***z*** = Run number.

Appendix 3 – Installation guide

To install the MOEA software, follow the instructions in the Readme files included in Appendix 2, and the instructions within each of the seven (7) Jupyter Notebooks.