Data Structure for Dynamic Geometric Data

Candidate Number: 086696

May 20, 2020

Abstract

There are a range of data structures available for the storage and efficient update of unconstrained sets of mutually non-dominating solutions. However there is little attention given to structures which are unbound, non-elitist and able to manage dynamic data, meaning a solution's objectives are re-evaluated over time. Objectives may be re-evaluated due to dynamic changes in the problem itself or due to estimates of objectives changing in noisy problems. A framework is proposed and an existing data structure is implemented for managing these relations between such dynamic solutions. A comprehensive empirical analysis is performed to identify the key characteristics and the scenarios under which the data structure can offer the best performance. Further avenues of work are recommended on how the data structure can be optimised for better use and efficiency gains potentially made.

I certify that all material in this document which is not my own work has been identified.

Contents

1 Introduction

A multi-objective optimization problem (MOOP) can be stated as a vector of decision variables which attempts to optimize a vector of objective functions whilst satisfying constraints [1]. The answer to this problem is the set of solutions that define the optimal trade-off between the competing objectives, which is referred to as the Pareto-optimal front.
Given two objective vectors, $\mathbf{v}$ and $\mathbf{u}$, $\mathbf{u}$ is said to *dominate* $\mathbf{v}$ iff $\mathbf{u}_d \leq \mathbf{v}_d$, $\forall$d = 1,...,D and $\mathbf{u} \neq \mathbf{v}$. This is denoted as $\mathbf{u} \prec \mathbf{v}$. A feasible solution lying on this Pareto-optimal front is non-dominated as it cannot improve upon any objective without degrading at least one of the others, therefore given the constraints of the model, it follows that no solutions may exist beyond this true Pareto-optimal front. The set of solutions which make up the Pareto-optimal front are *mutually non-dominating* [2], as demonstrated in Figure 1. Thus the goal of multi-objective algorithms is to locate the Pareto front that consists of these non-dominated solutions, and to also find a set of non-dominated solutions as diverse as possible so that the entire range of the Pareto-optimal front is represented [3] [4].

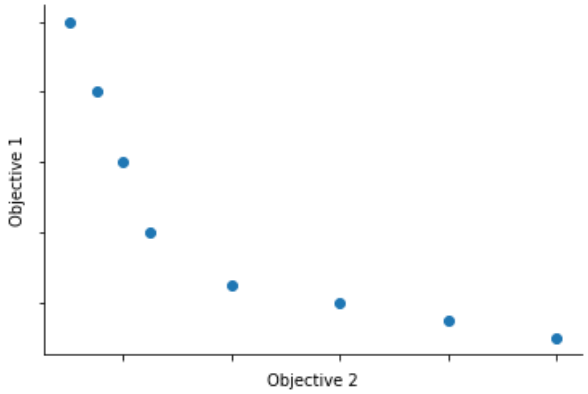

Figure 1: Example Pareto front

MOOPs have many real world applications across numerous disciplines. They are employed when there is a trade off between the best set of solutions, whether this be design parameters for an engineering problem or to aid in designing a neural network for an existing problem [5].
Evolutionary Algorithms (EAs) are a successful and widely used general problem solving method that mimic, in a simplified manner, biological evolution. In nature, individuals have to adapt to their environment in order to survive in a process called evolution, where features which make an individual more suited to compete are preserved when it reproduces, and other features which make it weaker are eliminated. Over subsequent generations the fittest individuals survive, with their best features being passed on to their descendants during the sexual recombination process known as crossover [6]. As the number of generations increases, the overall fitness of the population should also increase.
Multi-objective evolutionary algorithms (MOEAs) maintain a population of solutions, while dealing simultaneously with a set of potential solutions as they try to converge towards the Pareto-optimal front. It is common in evolutionary multi-objective optimization to keep track of the non-dominated individuals by storing them in a special data structure called an archive [7]. An archive of non-dominated solutions is a data structure for keeping track of the known non-dominated points $\{x^{(1)}, ..., x^{(n)}\}$ belonging to `X` of a MOOP [8].
The archive is updated during the optimization process whenever an individual in the population changes therefore the computational cost of updating the archive is critical [1] [9].
The aim of this project is to implement the guardian dominator approach [2] which stores the Pareto dominance relations between objective vectors in an unbounded dynamic non-elitist archive. The key challenges faced by this approach are maintaining the set of dominance relations whenever a new solution is added, or when an existing solutions objective values change, due to the problem being dynamic in nature.

1.1 Literature Review Summary

The existing literature reviewed as part of this project looked into a range of data structures used to represent such archives. It also investigated the dynamic, noisy and robustness qualities of data.

1.1.1 Elitist MOEAs

Elitist MOEAs use external archives to store the non-dominated solutions of each generation, thus the archive contains the best estimate of the Pareto-optimal front at any stage [1] [10]. The final output of an elitist MOEA is the set of mutually non-dominated solutions stored in the final archive, however this set remains an approximation of the Pareto-set [11]. A non-elitist algorithm will also store dominated solutions as part of its archive. An archive is considered bounded if the number of points it can store is limited. This is a frequently used approach allowing reduction of the time needed to update a Pareto archive [7].

1.1.2 Bounded Archives

In most MOEAs, the truncation procedure on an archive is used to maintain the diversity of solutions, preserving the boundary individuals and deleting the individuals with the highest density. Having too many non-dominated solutions may reduce selection pressure and slow down the search [12]. It is called *overnondomination* when a large proportion of the population becomes non-dominated. This is considered disadvantageous [9]. However truncation leads to retreatment/oscillation, deterioration of quality, and slowing the rate of convergence, as it can potentially discard non-dominated points.
Forming an archive of all non-dominated solutions will mean that the archive always moves towards the true Pareto front as it cannot retreat or oscillate, allowing for sensible convergence criteria to be defined. When the frontal set contains all the currently found non-dominated objective vectors, it will become very large as the search progresses, and as it has to be queried at each generation, it can quickly become costly to maintain [10] [13]. A bounded archive will always have lesser quality than an unbounded archive, hence if an unbounded archive can be efficiently managed and updated, it is advantageous to use [14].

1.1.3 DMOOPS

Optimisation problems with at least two objectives in conflict with one another, and where either the objectives or constraints (or both) change over time, are referred to as dynamic multi-objective optimisation problems (DMOOPs). Solving DMOOPS requires that the Pareto-optimal set at different moments can be accurately found [15].
The goal should be to track a solution's movement through the solution and time spaces as closely as possible. The building blocks for solving dynamic optimisation problems are: maintenance of diversity so that the population can adapt more easily to changes; react to changes such as performing a hypermutation; use of memory to help preserve good solutions from the past (only useful when what occurs now can happen in the future); and having multiple populations as a way of increasing diversity [16].
At the start of the evolutionary algorithm run it is desired to determine both the dominated and non-dominated individuals, store these in the archive and then update the archives contents as the population changes [9]. The population will update after each generation, meaning the Pareto-optimal set is likely to have changed, therefore the dominance relations between vectors will have also changed, thus the archive will require an update. An update has two key operations: insert, in which a new solution is compared to the existing contents of the archive to check if it is non-dominated [1]; and edit, in which a solution has moved position in the archive, either becoming dominated or non-dominated.

1.1.4 Linear List Approach

In most MOEAs with elitism, the population is stored within a linear list due to its simplicity in implementation and use. In this structure, a new point is compared to all points in the list until a

covering point is found or all points are checked. The point is only added if it is non-dominated with respect to all points in the list. In the worst case, the whole list is browsed before adding a point. Any newly non-dominated points are appended to the end of the list. The complexity, in terms of the number of points comparison, is thus $O(N)$ where N is the size of the list [7] [8] [14] [17].
The basic linear list approach can be improved upon by reducing the number of dominance comparisons by trying to infer domination relationships using the transitive nature of Pareto dominance. Relations can be deduced from existing relations, such that if `A` dominates `B` and `B` dominates `C` then we can deduce that `A` also dominates `C`. These algorithms achieve a very crucial speedup, but they are specifically designed for populations where the domination relationship between individuals is relatively common. This assumption does not hold for problems with a large number of objectives. The fewer domination relationships there are, the fewer such relationships can be inferred, and in turn, the performance suffers. These methods do constitute a significant innovation since they are dynamic in that when one individual changes, the information about the non-dominated fronts can be efficiently updated [2] [9] [17].

1.1.5 Guardian Dominator Approach

The proposed data structure is based on the guardian dominator approach [2], which employs previously reviewed techniques to store the domination relations between vectors in an unbounded dynamic non-elitist archive. The main idea is that each member of the Pareto-set becomes a guardian of the solution it dominates in an effort to prevent keeping track of all domination relations between members of said archive. Attempts are made to store the minimum number of links to appropriately place new solutions into the archive. How a guardian is assigned is dependent on the parameters given to the archive. Some options only store the elements which the Pareto-set dominates, whereas others attempt to create chains between dominated solutions. This data structure is expected to perform especially well when objective values rapidly change as it allows for quick updates.

1.2 Project Specification Summary

This project's aim was to develop a package in Java that would implement a data structure suitable for playing the role of an unbounded dynamic non-elitist archive in a MOEA. It should be able to efficiently maintain the mutually non-dominating set when the objective values change over time, in that at any time step, the current estimate of the Pareto-optimal front can be retrieved.
The basis of the implementation of the data structure is detailed in [2], which describes an algorithm that uses the premise of guardian dominators to determine dominance relations between members of the archive.
The data structure described in [2] already has an implementation in MATLAB, however a Java implementation has been specially highlighted to be useful to the community that deals with the type of problems where such a data structure would be beneficial. An object oriented approach using references between the guards and guarding items is also envisioned to be quicker for larger populations.

1.2.1 Non Functional Requirements

The desirable, non functional attributes of the data structure describe aspects and features which are not essential to its operation, however will be beneficiary in making the data structure viable as part of a feature rich package. Therefore it follows that the archive should be thread safe in order for it to function correctly during simultaneous execution, allowing multiple threads to access the same shared data, but only allowing one thread to modify its content at any given time. It was originally outlined in the previous project specification that the data structure should be multi-threaded, however this was very much an oversight as the actions of adding and editing solutions are inherently atomic thus the only place multi-threading could be employed is when multiple solutions are requested to be operated on in a batch.
Dynamic, noisy, and robust data should all be able to be suitably managed by the archive in order to improve the range of problem types for which the archive is suitable. This can be done by detecting

duplicates when adding and editing solutions.
It was also desired that the Pareto-optimal set be diverse. This will be accomplished by the archive being unbounded and non elitist as the absence of truncation means no solutions will be discarded. If no solutions are discarded, the diversity of the elite set will not be reduced.
In addition to the concrete implementation developed, an interface has been proposed for such a data structure, as this will allow others to use, build on, and improve the provided implementation. An interface allows for the accurate description of the role the data structure will play, and what use cases it will be able to handle. It will also allow a set of unit tests to be created, which all future implementations must conform under, to ensure the correct behaviour is performed under usage scenarios.

1.2.2 Evaluation

An empirical analysis of the data structure was performed to evaluate the performance characteristics under a range of varying use types and protocols, in order to identify how best to configure the parameters of the archive, as well as how best to investigate new extensions or current bottlenecks of the concrete implementation. Particular focus was given to the method of guardian assignment at each stage. The overarching aim of this project was to develop an implementation of a data structure that would offer significant improvements in performance and efficiency over the standard linear list approach for handling when objective values change in a MOOP. This was the hypothesis that was being tested, and in order for this to be considered proved, the following set of functional requirements would have be met with sufficient performance.

1. Archive is unbounded.
2. Archive is non-elitist.
3. Be able to evaluate new locations.
4. Be able to change the objective values of a dominated member of the archive.
5. Be able to change the objective values of a non-dominated member of the archive.
6. Be able to retrieve the non-dominated set at the current time step.
7. Perform in better runtime than a linear list approach.

To achieve these criteria, the project was split into three stages: design and planning, development, and analysis of results. Once these phases were completed, the criteria could be evaluated. If all the functional requirements were met, then the hypothesis could be considered proved.

2 Design

The first stage in achieving the evaluation criteria was to create a design of the system which could be used as a blue print to follow throughout the development of the archive. Correct thorough planning would allow for the evaluation criteria to stay at the core of the project, averting the risk of losing scope of the project objectives.

2.1 Application Programming Interface

The design process began by deciding what would be offered in the public application programming interface (API). This would determine what features would be offered to end users of the package. The creation of an interface will allow for multiple implementations of a dynamic archive to be developed in the future. This also means that one set of unit tests can be published against the interface, but with multiple implementations being able to be tested from them, thus reducing development time of future implementations and increasing consistency amongst implementations.
The interface supplied provides a general framework for dynamic unbounded non-elitist archives of which a `GuardianArchive` has been implemented as part of this paper. A dynamic solution abstract

class is provided which contains a property listener attached to the objective values to evaluate, which will trigger an archive update upon any changes.
The key methods in the interface are `add` and `editObjectives`. These will be called when an update to the archive will need to be triggered as there are new objective values. There is the option of adding multiple solutions at once, however this will treat each addition as a single atomic action, sequentially adding them to the population. Further functionality of the API is checking that the population contains either a single solution, or a collection of solutions, regardless of whether or not the solution is non-dominated. Additionally, there is also the functionality to retrieve the current non-dominated set as an iterable, and to check whether an individual solution is a member of the non-dominated set. The size of the Pareto set and the size of the entire population can also be retrieved. The API does not offer the functionality to delete solutions from the population, nor does it offer functionality to forcefully add solutions to the Pareto set without any checks as this would disrupt the internal logic of dominance relations between solutions.
Once the public `DynamicArchive` API was created, the `GuardianArchive` was implemented. It is parameterized to contain `GuardianSolution` or any of its sub-classes. `GuardianSolution` is a sub class of `DynamicSolution`, and they are employed to store the direct relations between individual objective vectors. They offer the package private functionality of each solution being allowed one parent solution (guardian dominator); and a list of child solutions. As new solutions are added, or the objective values of current solutions change, the child-parent relations between solutions will be updated as the relations represent the guardian dominance between solutions. This forms the tree structure shown in Figure 2.

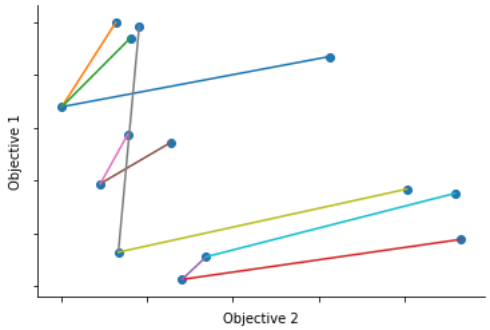

Figure 2: The internal tree structure of the archive which is created when solutions are assigned guardians due to their objective vectors being added or changed.

2.2 MOEA Integration

The existing Java library, the MOEA framework [18], has been extended in order to increase use-ability and accessibility of the package to developers. This has been done by extending the existing `Solution` class so that data will conform to the same standard used by existing archive implementations. Their Pareto dominance comparator has been used to check the dominance relations between two solutions as this is tested and optimised.

2.3 Data Representation

The data will be stored in the class `GuardianSolution` which extends the abstract class `DynamicSolution`. This in turn extends the `Solution` class found in the MOEA framework. The `Solution` class stores the variables, objectives, constraints, and any attributes the solution may have. The objective vector is stored as a primitive array. If a user wishes to create a new solution, they can do so by either passing an existing solution to a `GuardianSolution` constructor or by initialising a `GuardianSolution` with a one-dimensional array of doubles. The `DynamicSolution` class builds on this by providing the framework for triggering an update to the archive through the use of a property change listener which watches the `objectives` field for any changes.
The `GuardianSolution` instances are encapsulated by the `GuardianArchive` class which extends

`DynamicArchive`.
The list of solutions which make up the non-dominated set are also the roots to each tree in the forest, thus the archive need only store these solutions as the remainder of the archive can be inferred from the guardian relations. Every dominated solution must have a guardian, therefore is in turn a member of a tree.
The `GuardianArchive` class manages the relations between the solutions, ensuring that every parent solution dominates all of its children. By storing solutions in a tree structure, it allows the search space to be reduced when comparing dominance, as any solution stored in a level lower in the tree than a dominated solution will also be dominated, due to the transitive nature of dominance as described in Section 1.1. A level order traversal of the tree has been used to search for solutions, because likely guardian candidates will be closer to the root as the population evolves, thus reducing the search space and in turn the number of domination comparisons. The concluding design of the package can be seen in figure 3.

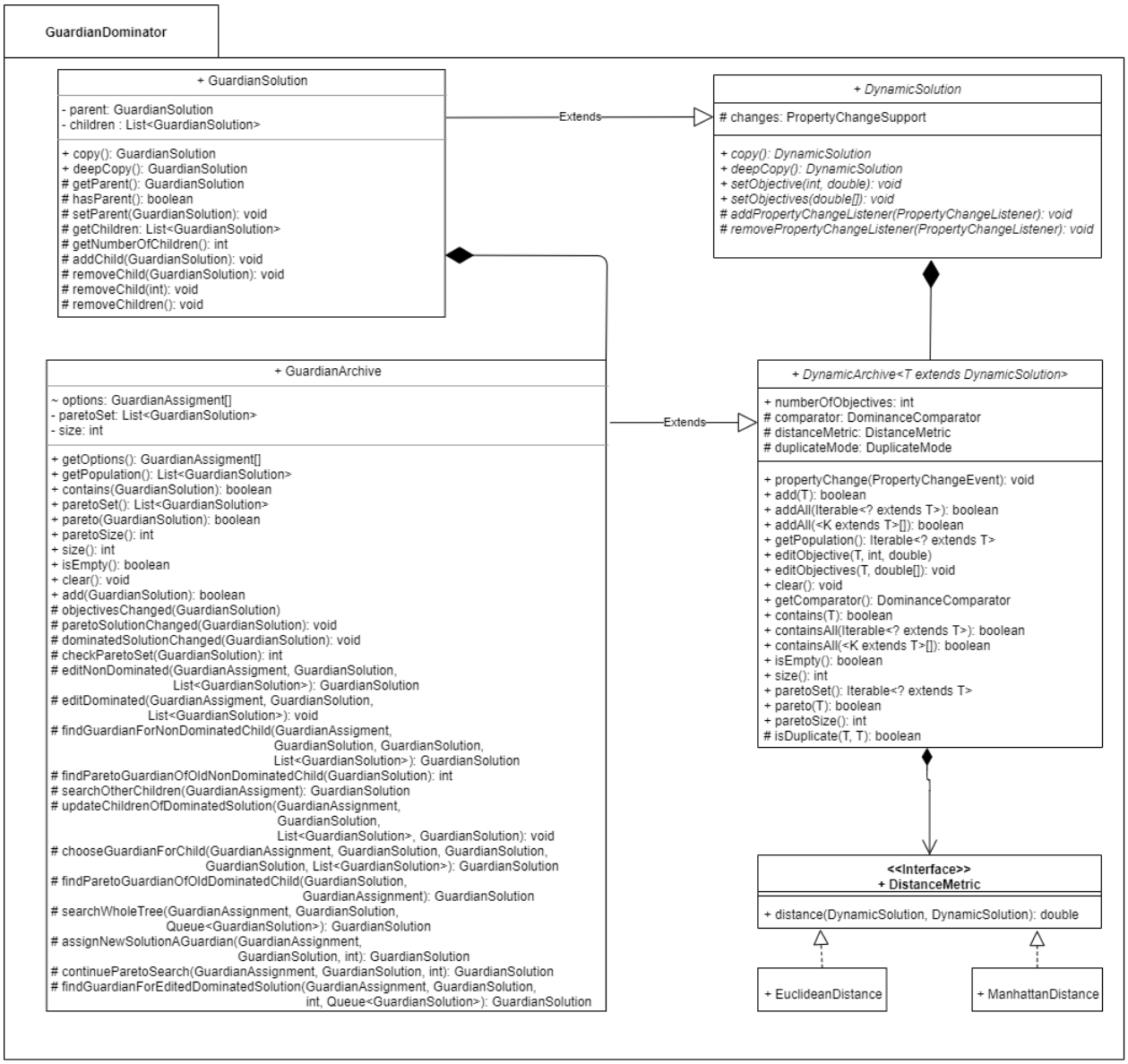

Figure 3: UML model of the guardian archive

2.4 Performance

The project requirements specify sufficient performance which is acknowledged to be ambiguous here. There is no baseline implementation to draw a comparison to as there has been very little research into

unconstrained non-elitist dynamic archives. It is expected there will be worse performance compared to an elitist archive due to the larger population size, and there not being an ordering of Pareto solutions as is common in elitist archives.
To maximise performance, the implementation takes advantage of caching to reduce computation throughout. For loops are primarily indexed based to avoid the overhead of using an iterator. No maximum or minimum memory requirements were specified, as this will very much depend on the end users system architecture, therefore we do not want to preemptively limit performance by reducing the memory footprint at the expense of computation.

2.5 Optimisation

Naturally, we want to minimise the number of domination comparisons required at each time step therefore it follows that no two solutions should be compared more than once per add or edit operation. To achieve this, when comparing a solution against the current elite set, the index is recorded of the first member to dominate the newly added or edited solution. This index is only revisited if the solution requires tutelage from the elite set as no other guardian is available.
The archive favours guardians which are deeper in the tree as this will reduce the search space of future operations due to the transitive nature of dominance. For example, when assigning a guardian based on which dominating solution guards the fewest from a set, the elite set will only be searched for a guardian if there has not been a dominating member from the set so far, with the count of how many solutions a candidate guards being reset for each component that makes up that set. This selection technique was chosen as there may be a member of the elite set which guards fewer solutions, however the solution which guards the fewest from a tree is a better candidate as it allows for the future search space to be reduced, owing to the level order search of a tree having the stopping condition of not to continue if solutions are no longer dominating. This relation also holds true for when a guardian is assigned based on closest proximity.
The Euclidean distance has been used here to determine proximity, however other metrics such as the Manhattan distance may be more appropriate, depending on the nature of the problem.

2.6 Error and Exception Handling

Methods return a boolean value based on if the operation was successful. The solutions have been sanitised for either being null or having an incorrect number of objectives. The constructor of the `GuardianArchive` takes a `Duplicate Mode` enum as a parameter, which can configure the archive to either: `ALLOW_DUPLICATES`, `NO_DUPLICATE_OBJECTIVES`, or `ALLOW_DUPLICATE_OBJECTIVES`. In the case of `ALLOW_DUPLICATE_OBJECTIVES`, it is the decision variables which are not allowed to be identical whereas the objective values can be. When `ALLOW_DUPLICATES` is chosen, both the decision and objective values can be identical.
Scope is obeyed in that the user cannot access methods which affect the internal relations between nodes. These have been kept package private in order to protect the internal forest structure from external interference.

3 Development Process

Once the design was complete, next began the development process by following a rigorous outlined methodology, which has been thought out for which technologies to use and how to adapt to any changes in design.

3.1 Methodology

The nature of this project is primarily research based, as it investigates the benefits and drawbacks of using a guardian dominator archive implementation. The development methodology followed was an agile Kanban approach where the backlog of tasks was continuously updated as the requirements for each stage became more refined. The advantage of using an agile approach over the Waterfall

approach was that the product requirements are able to adapt over time, which is of particular use when a problems complexity has not been fully understood [19].
The use of the methodology varied from this in that a weekly deadline was worked towards a meeting with a supervisor, where the previous weeks progress could be reviewed and the backlog of remaining tasks could be refined. My supervisor was in part acting as the product owner through the guidance they gave in regards to what features should be included and their format.

3.2 Technology Stack

The development process began by setting up a development environment. IntelliJ was elected to be used for its rich range of features, such as its debugger and profiler. The project uses the build tool gradle. The library JUnit 5 was chosen as the unit testing framework as it is the latest release, and the most popular. The version control system employed was Git, as is industry standard. The use of Git allowed for features to be pushed to the main branch once code changes passed the unit tests. The remote repository can be found at https://github.com/lhs211/Dynamic-Geometric-Data.

3.3 Development

Development work on the data structure began by writing a comprehensive set of unit tests against the API. This would ensure that all code changes made would not add any feature breaking bugs to the package. As the package grew in functionality, the code coverage of the tests unfortunately decreased as new scenarios were being run which had not been properly accounted for by the test suite. This led to two helper test methods that accessed the package's internal dominance structure using reflection, being written. The methods `checkGuardianDominates`, and `writeDominationStructure` were especially beneficial as they conveyed when a dominated solution had not been assigned a parent, or when a cycle, in place of a tree, had been created. These were the two most common frustrating scenarios during the debugging process, occurring frequently when a solution had been misassigned a guardian.

4 Empirical Analysis

The empirical analysis carried out aims to identify the performance characteristics of the data structure and how best to maximise these based on the problem type and guardian assignment technique.

4.1 Experiments

All empirical work was conducted on a laptop running Windows 10 with the machine specification: Laptop Quad-core 2.8 GHz CPU. L1d cache: 128KB, L1i cache: 128KB, L2 cache 1MB 16GB Ram @ 2400MHz. All timings have been recorded by measuring the CPU time dedicated to the execution thread when interacting with the data structure excluding all other time costs, such as sampling from the analytical distribution.
The first set of experiments use the protocol employed in [8] for generation of objective vectors from a controlled analytical distribution, which removes the stochastic element of an optimiser from the results. The archive is constructed from a sequence of D dominated (d > 0) and N non-dominated (d=0) normally distributed objective vectors according to:

$$y^{(k)} \sim \mathcal{N}\left(\frac{d(N+D)}{k}\mathbf{1}, \mathbb{I} - \frac{1}{m}\mathbf{1}\mathbf{1}^T\right) \tag{1}$$

where $\mathbb{I} \in \mathbb{R}^{m\times m}$ is the identity matrix and $\mathbf{1} = (1, 1, ..., 1)^T \in \mathbb{R}^m$ is the vector of all ones. The distribution has unit variance in the subspace orthogonal to the $\mathbf{1}$ vector. The parameter d controls the systematic improvement of points. At position k of the sequence it is assigned a value of 0 with a probability $c\frac{N'}{N'+D'}$, where D' and N' denote the number of remaining dominated and non-dominated points to be placed into the sequence, otherwise d is assigned a value > 0. For $c < 1$ there is a preference for seeing more dominated points later in the sequence, and for c > 1 there is a preference

for seeing more dominated points earlier in the sequence. The non-zero value employed for d is not specified in [8], hence we use $1/N$ here.
In the simulation, for each member of the population an underlying 'true' objective location is stored. This 'true' location is the mean as sampled from Equation 1. An evaluation of a new solution is sampled from the mean at time k, giving the 'observed' noisy objective vector. When a solution is re-evaluated, another objective vector is drawn from the sample, and the y of a solution is taken as the mean average of the noisy objective vector samples taken so far. This allows for objective vectors to have the refinement property as seen in noisy optimisation. In order for the desired values of N and D to be reached, each point must be evaluated numerous times before it reaches an approximation to its mean value, thus its' desired location. As points are selected randomly from either the dominated, or the non-dominated, set it is not guaranteed that points will converge to their mean objective values. Therefore, two approaches have been taken to sampling.

1. The selected objective vector's y is updated to its true location.
2. The selected objective vector is sampled once, with its y being the mean average of samples taken from that vector so far.

The two iterative generating processes of a population being examined are the generation of new solutions according to Equation 1 where the number of non-dominated solutions is a parameter, and then the selection of which member of the population to change:

1. A random member of the population is changed.
2. An elite member of the population is changed.

This creates four distinct simulations when combining the sampling technique with a selection procedure. All four simulations have new solutions generated from the distribution that have been sampled once, hence they contain the desired multivariate isotropic Gaussian noise. A solution is chosen using a selection method and its value is updated using a sampling method. This leads to the following simulations:

S_1: Noisily distributed new solutions, randomly changed true objective solutions.

S_2: Noisily distributed new solutions, elite randomly changed true objective solutions.

S_3: Noisily distributed new solutions, randomly changed converging solutions.

S_4: Noisily distributed new solutions, elite randomly changed converging solutions.

Each simulation was run for the values of $c = \{0.5, 1.0, 1.5\}$. The situation where there are lots of dominated points later in a sequence, then proportionally more non-dominated points later, is especially interesting as this will demonstrate how larger, longer chaining trees will behave. Each simulation was also run for values of $m = \{2, 5, 10\}$ to analyse how the number of objectives would affect the archive update procedure. Finally, each simulation was run with a true elite set size of {128, 512, 2048}, as this will expose the tree structure's handling of non-dominated fronts that are different in size. All of these conditions are tested for the guardian assignment methods detailed in Table 1.

4.2 Guardian Assignment Technique

The configurations in Table 1 were chosen as they cover a wide range of the methods described in [2]. Combinations C_{1-3} vary only in their choice of which member of the elite set should become a guardian when a new solution is added. The remaining five options all give preference for choosing an immediate guardian without further comparison and if that's not an option, they give preference for solutions which are the first dominating members of the corresponding set.
Combinations C_{4-6} take a consistent approach where the same assignment method is used at each option. This may not be considered optimal for all options, as sometimes the solution which is closest, or guards the fewest others, may also be the same solution that is chosen through the greedy option one approach. That method of assignment takes far less comparisons as its search stops once a potential guardian is found, rather than searching for the highest quality guardian from each set.

Label	Option combinations	Description of guardian assignment
C_1	1.1, 2.1, 3.1, 4.1, 5.1, 6.1	Assigns first dominating element to new solution, and first dominating to changed solution
C_2	1.2, 2.1, 3.1, 4.1, 5.1, 6.1	Assigns closest elite to new solution and first dominating to changed solution
C_3	1.3, 2.1, 3.1, 4.1, 5.1, 6.1	Assigns dominating elite which guards the fewest to new solution, and first dominating to changed solution
C_4	1.1, 2.2, 3.1, 4.2, 5.2, 6.2	Assigns first dominating found - always searching through the changed sub tree first for the changed solution
C_5	1.2, 2.3, 3.2, 4.3, 5.3, 6.3	Assign closest dominating solution from comparison sets
C_6	1.3, 2.4, 3.3, 4.4, 5.4, 6.4	Assigns dominating solution with fewest guarded solutions from comparison sets

Table 1: Method combinations tested empirically. Options refer to Algorithms 1 and 2 in [2]

4.3 Empirical Results

The simulations were run for 100,000 time steps using the option configurations detailed in Table 1. The procedure followed is that, at alternate time steps, either a new solution is added to the population, or an existing member of the population has its objective vector changed. The time taken to perform the operation, the number of comparisons required, and the size of the elite archive were all recorded at each time step hence add and edit operations can be compared independently.

4.3.1 Timings

Figure 4 shows how a population with solutions spread evenly throughout its lifespan performs. The growth follows a more discrete pattern as there are a multitude of operations being performed, not all of which are equal in cost thus causing sudden spikes in growth.

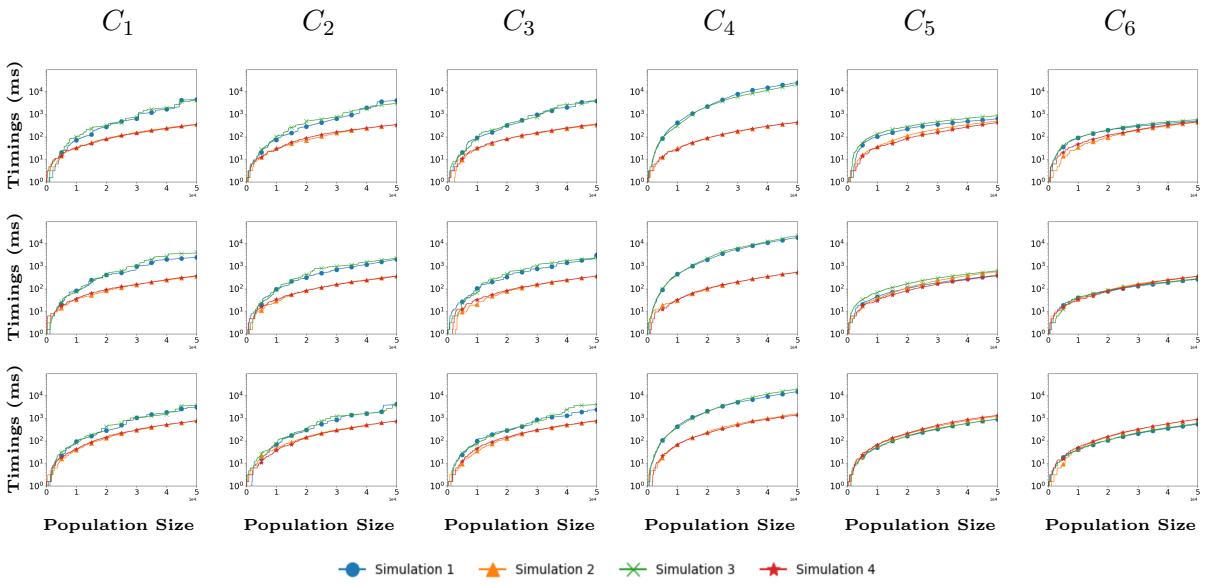

Figure 4: Mean time taken to update data structure per operation. The logarithmic scale being used shows how the cumulative mean time taken to update the archive increases as the archive becomes larger. Results for population, c = 1.0**. Top row:** Elite = 128**. Middle row:** Elite = 512**. Bottom row:** Elite = 2048**.**

When comparing the different guardian assignment techniques in Figure 7, we can see that for all four simulations, C_5 and C_6 perform the best as their rate of growth is lowest for all plots. S_2 and

S_4 have a consistent growth rate across all plots, showing the guardian assignment method has little effect on the overall time taken to update the archive. This does not hold for S_1 and S_3. These two simulations edit members from the entire population, rather than just the elite set, therefore are slower to converge towards the true Pareto front. The growth curves appear to follow a more discrete nature suggesting large jumps are made in place of continuous growth. This may be down to the emergence of more non dominated solutions. This is in sharp contrast to C_4 which sees a higher rate of continuous growth for S_1 and S_3, and shows less signs of slowing down. This increased time taken compared to S_2 and S_4 is down to elite members acting as the guardian dominator for orders of magnitude more solutions compared to when guardians are assigned using a greedy approach, therefore the first member of a set is likely to have a disproportionately large number of children due to the method of the implementation. This is supported by the same trend in C_{1-3}, except it is amplified for C_4 as a greedy approach is used at every option. For a true elite size of 128, it can be seen that S_1 and S_3 in populations C_5 and C_6 initially grow at a quicker rate, but resume the same growth rate as their counterparts by the end of the population. As the elite set increases in size, there is shown to be very little disparity between plots, however there does appear to be a smaller difference between the time taken at each population interval.

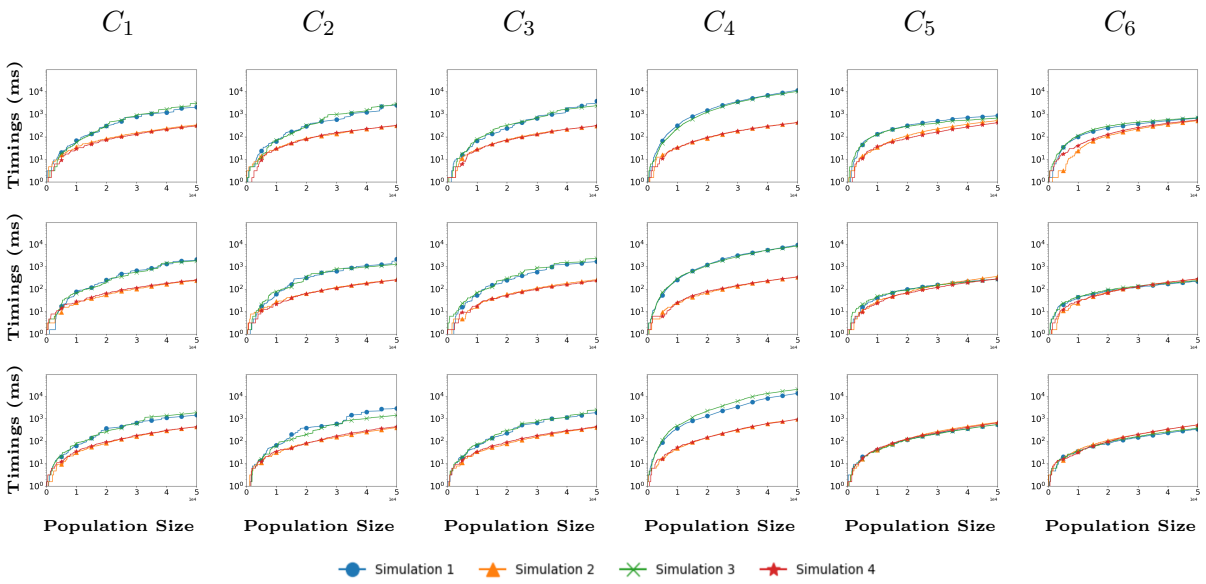

Figure 5: Mean time taken to update data structure per operation. The logarithmic scale being used shows how the cumulative mean time taken to update the archive increases as the archive becomes larger. Results for population, c = 0.5**. Top row:** Elite = 128**. Middle row:** Elite = 512**. Bottom row:** Elite = 2048**.**

When the proportion of non-dominated points is spread later throughout the sequence as in Figure 5, the rate of growth can be seen to be near identical showing that the data structure is resilient to when the true Pareto front appears in the sequence. This pattern is repeated when the non-dominated solutions appear earlier in the sequence as in Figure 6.
It is of note, that in C_5 the cost of calculating the Euclidean distance between solutions has not had an effect on timing, therefore the guardian assignment technique should not be discarded for fear of increased computation, when it can be seen to produce higher quality guardian assignments.

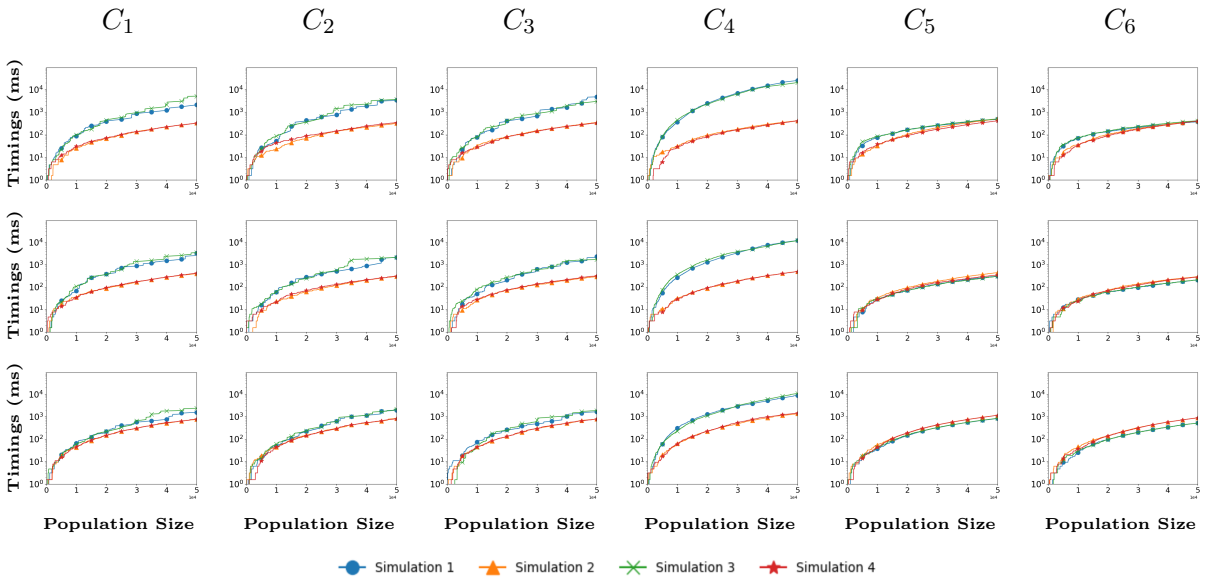

Figure 6: Mean time taken to update data structure per operation. The logarithmic scale being used shows how the cumulative mean time taken to update the archive increases as the archive becomes larger. Results for population, c = 1.5. **Top row:** Elite = 128. **Middle row:** Elite = 512. **Bottom row:** Elite = 2048.

4.3.2 Comparisons

It has been chosen to display the number of comparisons taken as this is the expected primary bottleneck to performance, which was supported when the data structure was analysed under a profiler. Only when the population grows in size, does the difference in comparisons become apparent. The growth follows a more discrete pattern as there are a multitude of operations being performed, some of which will take significantly more comparisons than others.

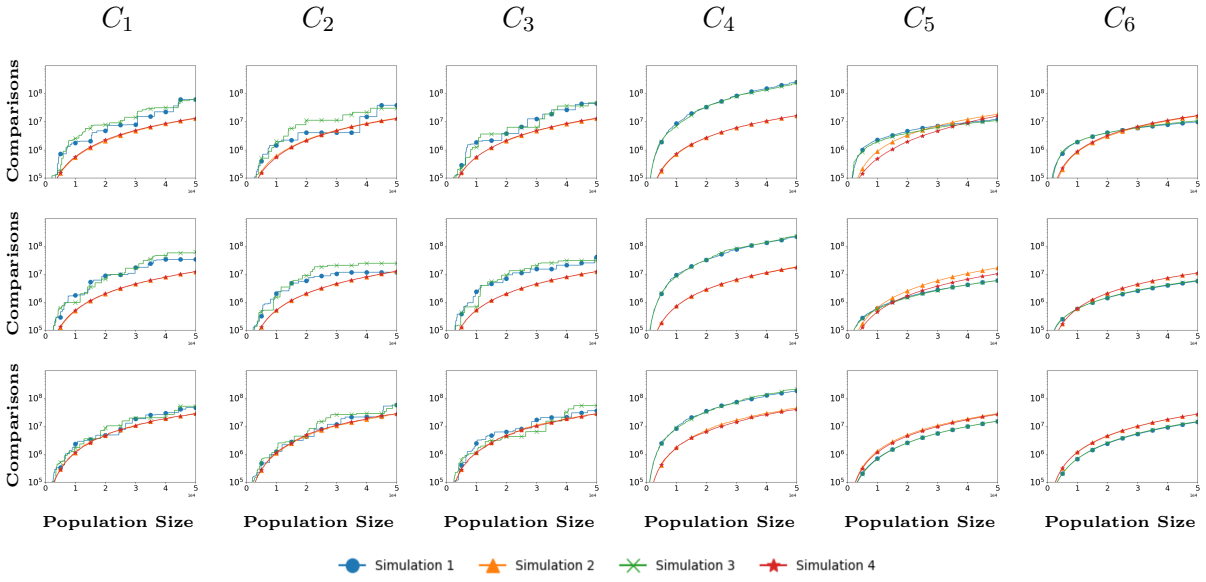

Figure 7: The cumulative number of comparisons required to update the data structure per operation. The logarithmic scale starts at $y = 10^6$ as the rate of growth is identical for all simulations and plots when the population size is small. Results for population, c = 1.0. **Top row:** Elite = 128. **Middle row:** Elite = 512. **Bottom row:** Elite = 2048.

Figures 8-9 show the cumulative number of domination comparisons required at each time step. Upon comparing the different guardian assignment techniques, we can see a clear distinction between the number of comparisons required for the different simulations. It is immediately clear that the

number of domination comparisons does not follow the time taken to perform an update equally. S_2 and S_4 have identical growth rates for $C_{1-3,6}$ whereas in C_4 the growth rate is marginally higher. S_2 is consistent with the other plots for C_5, unlike S_4 which actually performs better when there are 512 solutions in the elite set. This shows there is a consistency to performance when members of the elite set are regularly changed.
S_1 and S_3 follow a discrete growth pattern for C_{1-3}. This is likely down to their guardian assignment taking a greedy approach where possible, and choosing the guardian as an immediate predecessor if possible, which has a best case complexity of $O(1)$. This leads to a more linear growth trend which is also greater than in S_2 and S_4. This continues for C_4, which takes notably more dominance comparisons per time step than its counterparts. This trend is reversed however, in C_5 and C_6, where S_1 and S_3 give a noticeably lower growth rate. This difference is more apparent when the archive size is smaller. This increased performance is likely caused by a less concentrated number of children for members of the elite set, thus reducing the cost of an edit operation, when every child needs reassigning a guardian. These two simulations also edit members from the entire population, rather than just the elite set, and are therefore are slower to converge towards the true front.

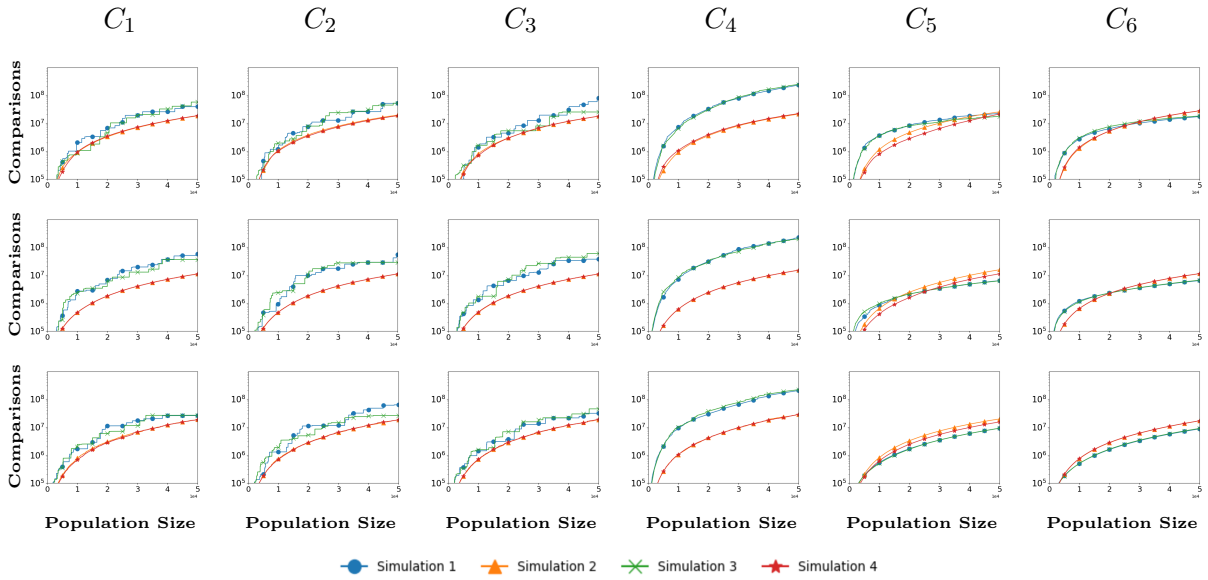

Figure 8: The cumulative number of comparisons required to update the data structure per operation. The logarithmic scale starts at $y = 10^6$ as the rate of growth is identical for all simulations and plots when the population size is small. Results for population, c **= 0.5. Top row:** Elite = 128**. Middle row:** Elite = 512**. Bottom row:** Elite = 2048**.**

When the proportion of non-dominated points is more heavily concentrated later in the sequence (Figure 8) compared to earlier (Figure 9), S_1 and S_3 for C_{1-3}, follow the trend of S_2 and S_4 more closely. This is because they only edit elite solutions, and once the elite set has appeared, this difference in simulation has been narrowed. This would explain why this difference is more apparent for the larger elite sets. This difference did not appear in the timings hence we can conclude that the distribution of points does not affect the overall performance of the data structure.

4.3.3 Elite Set

Figures 10-12 show the cumulative number of comparisons taken, when the elite set's size has changed after an edit operation. This was chosen to be graphed because the change in elite set is fundamental to the solving of a MOOP, thus understanding the characteristics of such an elite set is beneficial to being able to accurately utilise the data structure. If updating the elite set is significantly expensive, it would be inappropriate to use the archive on a problem which has a rapidly moving elite set. Markers have been placed every 50 data points. There is a difference in the frequency of markers per simulation due to whether the elite set is constructed primarily through add or edit operations as only data for edit operations are recorded here.

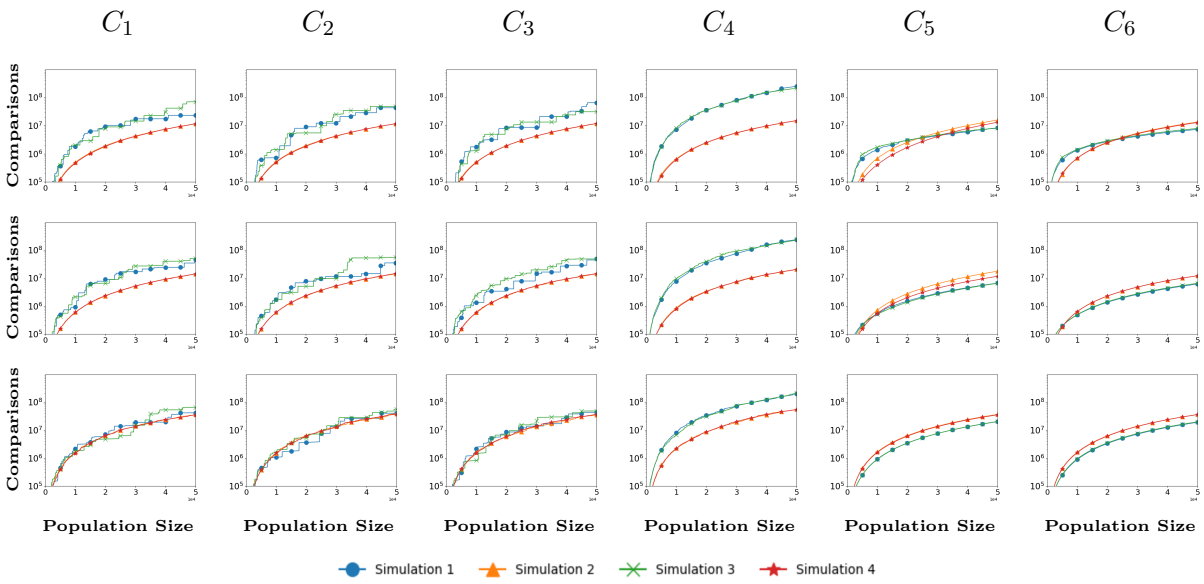

Figure 9: The cumulative number of comparisons required to update the data structure per operation. The logarithmic scale starts at $y = 10^6$ as the rate of growth is identical for all simulations and plots when the population size is small. Results for population, c = 1.5**. Top row:** Elite = 128 **. Middle row:** Elite = 512 **. Bottom row:** Elite = 2048**.**

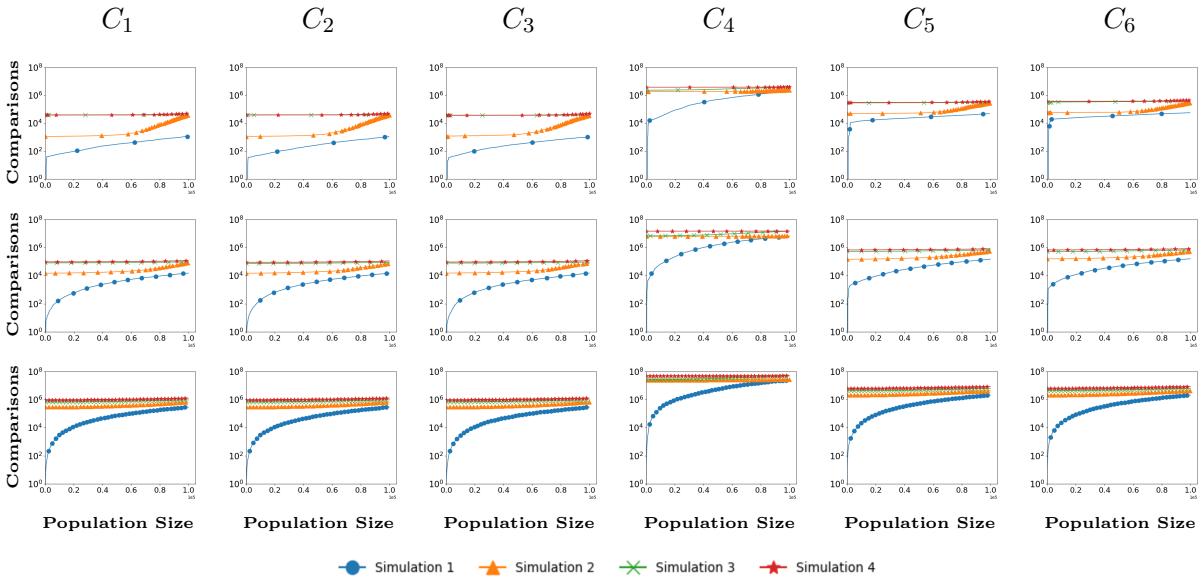

Figure 10: The ratio between the number of comparisons taken to update the elite set per edit operation. Results for population, c = 1.0**. Top row:** Elite = 128 **. Middle row:** Elite = 512 **. Bottom row:** Elite = 2048**.**

In figure 10, for S_1 where random solutions are changed to their true objective location, the number of comparisons starts to grow at a significantly lower rate than all simulations across all plots. It is also the best performing simulation, however it does have a tendency to reach the same rate of growth as the other simulations when the elite set is 2048. This is particularly noticeable for C_4. Overall the smaller 128 elite set performs best for this simulation as the change in growth tends to increase at a flat rate here, however this may be due to the decreased number of data points available. The guardian assignment C_1 produces the the least growth, however it is still outperformed by C_2 for an elite set of 128.

S_2 has the second best growth rate across all plots, which following the common trait from S_1 of moving directly to the true objective vector, shows that the archive performs better when solutions converge faster into the elite set. As the size of the elite set decreases, S_2 differs less from S_3 and S_4.

This may be down to larger elite sets not being as fast to converge. The trend for C_5 and C_6 is very similar. In the smaller of the elite sets, the rate of growth correlates stronger with the population size increase. C_{1-3} show the best performance overall across population sizes.
S_3 and S_4 follow a similar trend to each other across all plots. The rate of growth is always at a near constant rate. S_3 does have a tendency to minorly outperform S_4 which is consistent with S_1 outperforming S_2, however the difference is less refined. The size of the elite set does not have a noticeable impact on the performance of these simulations. The best performing guardian assignment method is C_1.

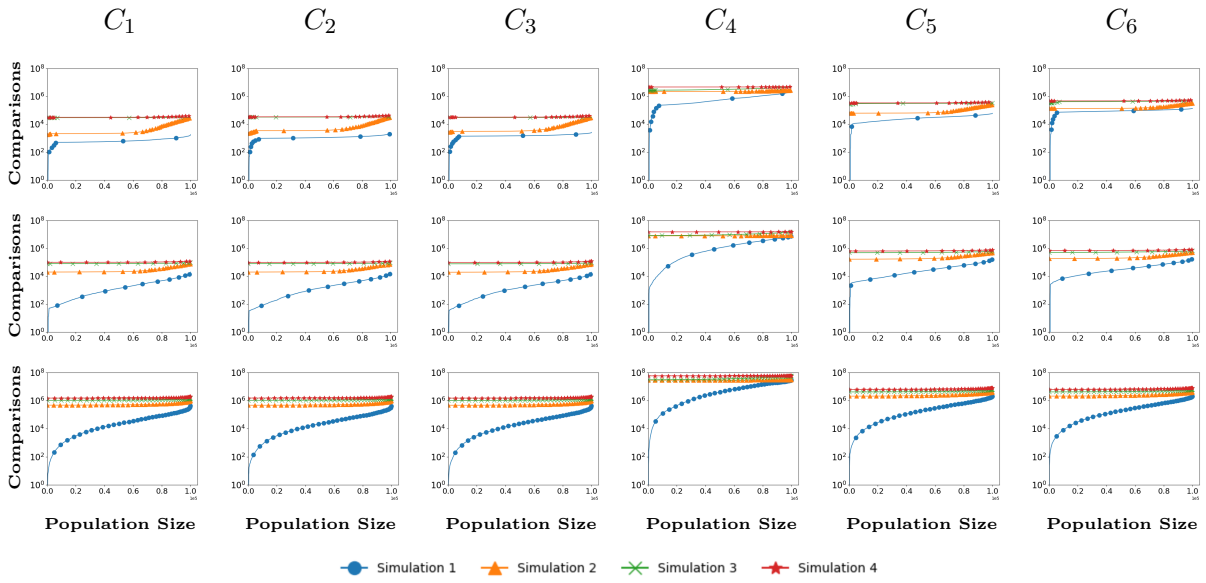

Figure 11: The ratio between the number of comparisons taken to update the elite set per edit operation. Results for population, c = 0.5**. Top row:** Elite = 128 **. Middle row:** Elite = 512 **. Bottom row:** Elite = 2048**.**

To investigate how the value of c affects the number of comparisons used to update the elite set, Figures 8 and 9 should be compared. When $c = 0.5$, non-dominated points had a higher probability of being generated later in the sequence, whereas when $c = 1.5$, non-dominated points have a tendency to be generated earlier in the sequence. This does not mean other points cannot become members of the elite set and then later be displaced out.
S_1 is the most noticeably different simulation. When $c = 0.5$ there is a slower rate of growth to begin with which sharply accelerates as the population begins to reach its maximum. This differs from when $c = 1.5$ which instead has a sharp immediate growth that starts to flatten to a constant rate once the predicted size of the elite archive is presumably met. The frequency of markers shows that the majority of markers occur earlier when $c = 1.5$ and later when $c = 0.5$. This correlates to distribution of points within the sequence. The total number of markers does appear to be consistent however, showing that this change in distribution is not caused by a change in the number of edit operations taking place.
S_2, S_3, and S_4 all appear to follow the same trend, where there is a reversed concentration of markers, but little to no change in rates of growth. The reversed concentration of markers also only tends to be prevalent in the larger elite sets as for 128 there does not seem to be a shift with the majority of edit operations taking place towards the end of the population's lifespan.
It is interesting that only one of the four simulations rate of growth was changed by the distribution of non-dominated points. S_1 randomly edits any member of the population, changing the objective vector to the true underlying location. This makes the true Pareto front likely to appear quickly, therefore when the true non-dominated locations appear later in the sequence, the existing front will move forward to the true front, and then the number of comparisons required to perform an update reduces.

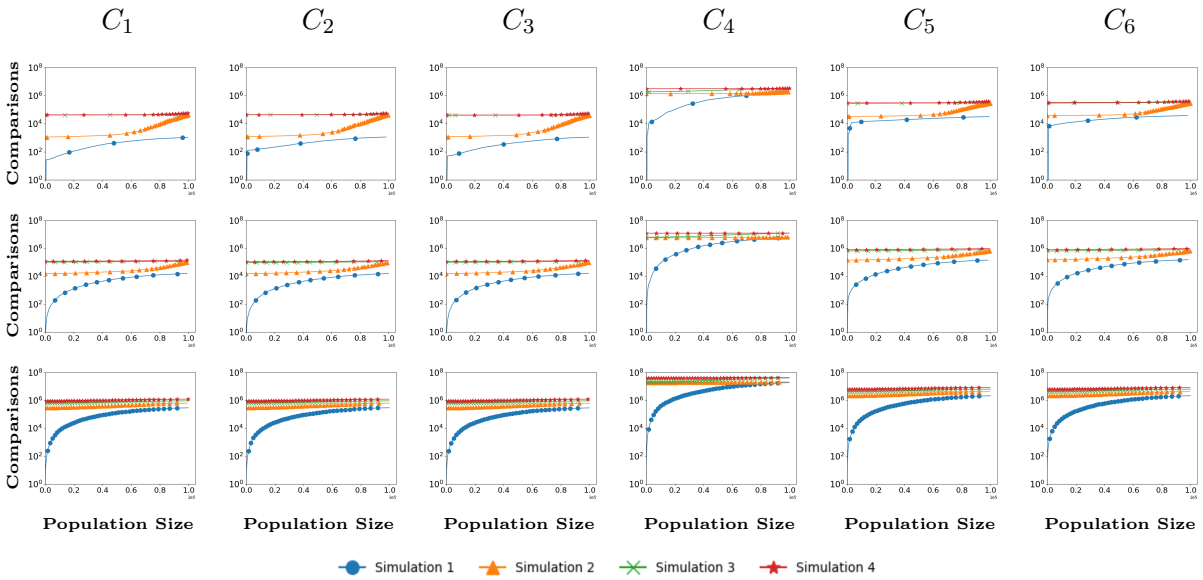

Figure 12: The ratio between the number of comparisons taken to update the elite set per edit operation. Results for population, c = 1.5**. Top row:** Elite = 128 **. Middle row:** Elite = 512 **. Bottom row:** Elite = 2048**.**

4.3.4 Number of Objectives

Figure 13 shows on a logarithmic scale how the number of comparisons required to update the elite set increases for populations with an increased number of objectives, $m = \{2, 5, 10\}$. The value c is 1.0 and the true size of the elite set is 512. This time the objective values are not isotropic as being so would mean the nature of the dominance relations would not change. The number of comparisons has been chosen to be compared, instead of timing, as naturally the length of the objective vector has increased thus there is more to compare. It is of interest here to determine if the Pareto dominance relations between solutions can still lead to a well formed tree structure.

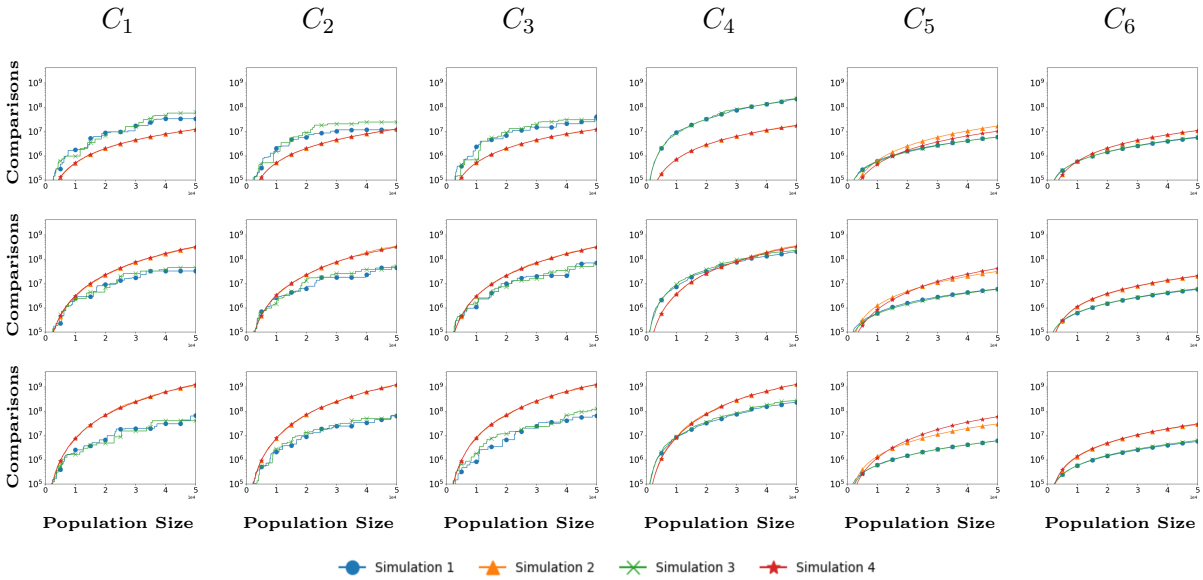

Figure 13: The cumulative time taken to update the data structure per operation. Top row: $m = 2$ **. Middle row:** $m = 5$ **. Bottom row:** $m = 10$**.**

The number of comparisons taken is shown to increase when the number of objectives is changed however the magnitude of the increase in comparisons differs between guardian assignment method. It can be seen that C_{1-4} are affected heavier by an increased number of objectives in comparison to C_{5-6}. This is likely because C_{5-6} distribute children between potential guardians better, thus

no one guardian will have significantly more children than any other. The increase in comparisons holds true for each guardian assignment method therefore it can be concluded that the archive is susceptible to changes in the number of objectives. This occurs because guardian assignment is built upon the transitive nature of Pareto dominance relations, which have a tendency to break down in higher dimensions as solutions tend to mutually dominate each other. This leads to shorter trees, which in turn leads to an increased search space thus an increased number of comparisons. This is particularly expensive when a single solution has a large number of children as they each will need reassigning a guardian.

4.3.5 Linear List Comparison

The evaluation criteria specify that the archive should out perform a standard linear list approach therefore a basic linear list implementation has been provided to act as a benchmark archive. It has been written under the same `DynamicArchive` interface, conforming to the same set of unit tests to ensure quality. Here the number of comparisons taken has been compared to show the efficiency gains made by the `GuardianArchive`.
Figure 14 demonstrates how the `GuardianArchive` performs in comparison to the linear list approach. The archive has two objectives and non dominated solutions distributed according to $c = 1.0$. C_1 has been chosen as the guardian assignment method for its consistency across previous plots and C_4 has been included to see if the worst performing `GuardianArchive` can still outperform a linear list.

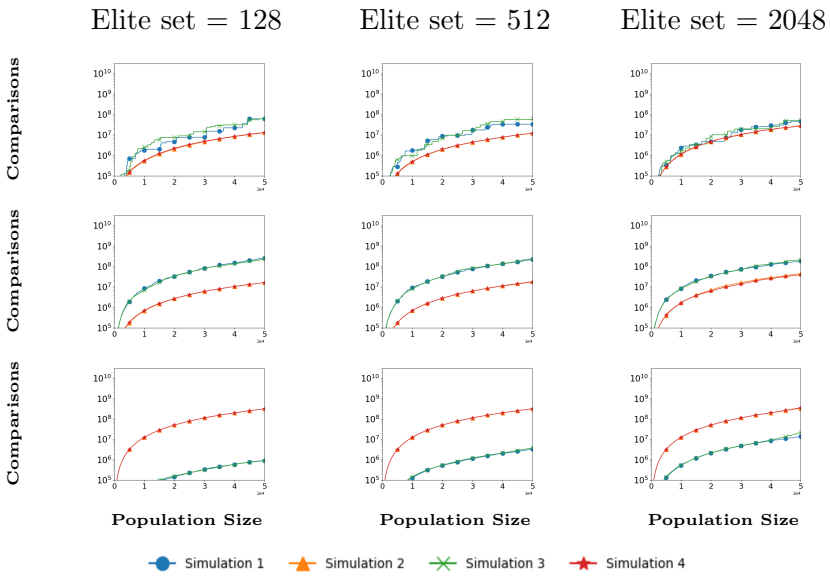

Figure 14: The cumulative number of comparisons taken to update the data structure per operation. Top row: C_1 **. Middle row:** C_4 **. Bottom row:** Linear List**.**

Figure 14 compares the number of comparisons taken by each archive to update the elite set. We can see that the linear list is significantly the worse performing archive for S_2 and S_4 as there is a far steeper rate of growth for a far higher number of comparisons. It is disappointing to see however that S_1 and S_3 perform remarkably better using a linear list approach. These two simulations edit solutions from the entire population, therefore we can see the cost of updating a dominated solution is far less without the presence of a guardian structure to maintain. This shows that a guardian archive is only advantageous when solutions are primarily edited from the current elite set.

4.4 Results Discussion

This study has proposed a suggested framework for the managing of an archive that allows the quick discovery of the non-dominated elite set from a constantly evolving population. It is able to handle both when new solutions are added to the population and when the objective values of existing

solutions change during the optimisation process which is of particular use in dynamic and noisy optimisation.
The archive's update mechanism relies on assigning a *guardian dominator* to every non-elite solution, with such relations being used to reduce the number of domination comparisons required.
The empirical results show that by assigning such guardian dominators, the location of the elite set in a constantly evolving population can be located quickly provided the right conditions are provided. The most prevalent condition to performance was which simulation was being ran. The simulations which randomly edited existing elite solutions (S_2 and S_4) had superior performance in terms of time, and number of comparisons taken as shown across Figures 4-8. The archive was shown to be resilient to changes in the distribution of non-dominated points as performance was not shown to deteriorate in Figures 5,6,8,9. Variation in the size of the elite archive was also not a considerable performance factor however there was a slight increase in the number of comparisons taken when the elite set was larger in size but this was not unexpected. The large elite set size of 2048 did not appear to cause problems of *overnondomination* which is promising. The most detrimental factor to performance was increasing the number of objectives as this reduced the number of dominance relations, thus expanding the number of children each guardian has. It would be ill advised to use this data structure for a problem with a large number of objectives.
Overall, C_4 was found to be the worse guardian assignment method as it was noticeably the worse performing across all conditions. There was no singular guardian assignment which proved dominant but the factors which affected the assignment method's effectiveness have been discovered. Ideally, children amongst potential guardians should be spread equally to avoid one solution having a large number of children because the cost of editing a dominated solution is high in comparison to the linear list which outperformed the archive in such scenarios, as shown in 14. The cost of distributing children equally however should be greater than the cost of unequally distributing solutions.
It is not determined whether the provided implementation's performance could further benefit from parallelisation, as multi-core architecture is prevalent in modern computing infrastructure, and utilising such has the potential to lead to significant performance improvements.

5 Critical Assessment Evaluation

This section outlines the scheme of work undertaken, offering a reflective look at how the project developed, justifying the decisions that were made and highlighting any errors that could be rectified through better planning and foresight. The results gathered are evaluated and potential future work is outlined.

5.1 Literature

The investigation of the existing literature on the role of the data structure in a MOOP showed this was a condensed, mostly unexplored area of MOEA which had little to none agreed, standardised approaches to how one should maintain an archive of solutions. The most common approach was to utilise a linear list which would take magnitudes more comparisons to update the elite set than a specialised data structure. It was found there were several approaches for storing bounded archives of elite static solutions, however optimisation problems which were dynamic in nature did not appear to have a specialised data structure besides the reference implementation [2]. The lack of existing literature in this precise area therefore meant that the linear list was the approach to better, with any improvements seen to be seriously advantageous.
The literature review could have been improved by taking a more refined look at the data structures themselves, rather than the role they play. This would have allowed for a greater variety of data structures to be reviewed, particularly focusing on such structures that are either dynamic or non elitist as this would have provided greater guidance when developing the aims and requirements.

5.2 Aims and Requirements

The aims and functional requirements outlined in the project specification can now be evaluated for the extent they are achieved and for their effectiveness.
The archive offers functionality to: evaluate new locations; change a dominated member; change a non-dominated member; retrieve the Pareto set at the current time step, and be unbounded, therefore the core evaluation alone would conclude that the project has been a success in meeting its goal, but this does not reflect quality. When evaluating the non-functional criteria, the quality of the archive can subsequently be inferred. From what was set out to be desired, the data structure conforms to a standardised interface, both for unbounded non-elitist dynamic archives, their dynamic solutions, and for the specific implementation of a `GuardianArchive`. It is able to sufficiently handle dynamic, robust, and noisy data by detecting duplicate decision and objective values, determined by a constant `EPS` value. The data structure is unfortunately not thread safe as this would have undermined the empirical analysis performed against it by allowing architecture to play a greater role in the timing characteristics. This would make comparison against other data structures complex if they too have not benefited from the dividends of parallelisation.
The desirable performance requirement was to outperform the basic linear list approach. The factor this would depend heavily on, was the guardian assignment technique used which was subsequently investigated by the empirical analysis. The analysis also cross examined the different scenarios under which the performance could be maximised. The reference linear list implementation provided demonstrated that the results gathered show a great reduction in the number of comparisons required to update the elite set under certain conditions. This criteria is thus only partly fulfilled as the archive does not always out perform the reference linear list implementation.

5.3 Process and Approach

The process used to develop the implementation proved to be effective as the benefits of an agile approach were reaped. Having a backlog of tasks that could be updated as problems became more refined proved to be beneficial in keeping the project aimed towards the specific problem at hand, rather than being led astray by other problems which some of the literature had focused on. The approach could perhaps have been strengthened further by rigorous, more significant deadlines, as the frequency of a weekly meeting did not always convey the urgency that work would need to be completed by in order for it to remain equally distributed amongst sprints. Improvements in efficiency could have also been planned from earlier on in the development process as the code base underwent several large refactors in order to reduce the number of comparisons. This was ultimately time consuming and inefficient.

5.4 Fitness for Purpose

The fitness for purpose of the provided library should be decided by the functionality, the results produced, and the performance given.
Firstly, the library provides all of the functional requirements expected of it, brought together under an interface. Secondly, the results produced were shown to be accurate, reliable and consistent between scenarios. The unit tests written are all passing without fail and the implementation has not thrown any exceptions hence the fitness for purpose is left to be evaluated by the empirical analysis.
The empirical analysis performed demonstrated that the archive is capable of performing numerous operations, under a reasonable rate of growth for both time taken, and the number of comparisons required. The results concluded that the archive is best suited for a fast moving elite set, provided that the correct guardian assignment method is chosen and the number of objectives is few. When comparing the performance to a linear list, it is clear that the number of comparisons taken to update the elite set is only far fewer when non dominated solutions are being re-evaluated. It considered that the performance of the data structure may vary with architecture, thus this is a limitation of the analysis provided. The extent of the analysis, although comprehensive, could of been greater with further guardian assignment techniques tested, as well as delving into dynamically changing the

guardian assignment technique as the population grows. Therefore the fitness for purpose could be further identified in greater detail than has been done so here.

5.5 Limitations

Despite the archives success in achieving its intended outcomes, there are some limitations to the functionality on offer. There is no ordering of the Pareto set itself which would be useful as an indicator of the quality of solutions. Currently, it would be down to the user to refine the elite set in order of favourability as a solution to the MOOP. Solutions could be sorted by the number of children they have where the more solutions indicates better quality as that solution is likely to dominate more of the dominated set. This would be a simple addition to add to the interface, with the bulk of the work being the consultation of existing frameworks to determine the best format to add this functionality.

5.6 Conclusion

In summary, the results proved successful, fulfilling the research hypothesis, and hopefully inspiring a future avenue of work into unbounded, non-elitist dynamic archives. The tree structure proved an effective method of representing and organising dominance relations between multiple solutions as it was able to significantly reduce the number of comparisons taken to update the estimate of the elite set, provided the population developed under favourable conditions, as demonstrated in the empirical analysis.
It is hoped that by providing a suitable data structure for a MOEA to use that the overall computation time to solve a MOOP will be significantly reduced. Although there are multiple alternative data structures, as demonstrated by the literature, use cases can vary dramatically and there is no one data-structure-fits-all approach hence it is imperative that a thorough empirical analysis is performed to identify if, and when, said data structure should be employed.

5.7 Future Work

The prevalence of parallel architecture in modern computing infrastructure should be a strong motivating factor for developing a thread safe implementation of the guardian archive. Thread safety would allow for multiple solutions to be operated on under one time step, thus reducing the bottleneck of having a long queue of solutions to operate on. It would perhaps be beneficial to consider the ordering of solutions before their objectives are evaluated as the order would strongly affect the amount of computation required.
Besides the addition of thread safety, performance could further be enhanced by an ordering of the elite set, and possibly the children of each guardian. This could be done when guardians are assigned based on which has the fewest children as solutions could be ordered by this. This will incur the increased cost of maintaining an ordering therefore the trade off will have to be investigated - a multi objective optimisation problem in its own right.
Furthermore, a comprehensive analysis could be undertaken on adapting the guardian assignment method, based on how the population is evolving. This would make the archive more resilient to differences between specific problems, hence giving it greater purpose as a general framework to use.

References

[1] N. Altwaijry and M. El Bachir Menai, "Data Structures in Multi-Objective Evolutionary Algorithms," *Journal of Computer Science and Technology*, vol. 27, no. 6, pp. 1197–1210, 2012.

[2] J. E. Fieldsend and R. M. Everson, "Efficiently identifying pareto solutions when objective values change," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 605–612. [Online]. Available: https://doi.org/10.1145/2576768.2598279

[3] J. E. Fieldsend, R. M. Everson, and S. Singh, "Using unconstrained elite archives for multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 305–323, 2003.

[4] K. Deb, *Multi-objective Optimisation Using Evolutionary Algorithms: An Introduction*. London: Springer London, 2011, pp. 3–34. [Online]. Available: https://doi.org/10.1007/978-0-85729-652-8_1

[5] T. Stewart, O. Bandte, H. Braun, N. Chakraborti, M. Ehrgott, M. Göbelt, Y. Jin, H. Nakayama, S. Poles, and D. D. Stefano, "Real-world applications of multiobjective optimization," in *Multiobjective Optimization: Interactive and Evolutionary Approaches*, J. Branke, K. Deb, K. Miettinen, and R. Słowiński, Eds., 2008, vol. 5252 L, pp. 285 – 327. [Online]. Available: http://epubs.surrey.ac.uk/831905/

[6] C. A. Coello Coello, "An introduction to evolutionary algorithms and their applications," in *Advanced Distributed Systems*, F. F. Ramos, V. Larios Rosillo, and H. Unger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 425–442.

[7] J. Yuen, S. Gao, M. Wagner, and F. Neumann, "An adaptive data structure for evolutionary multi-objective algorithms with unbounded archives," in *2012 IEEE Congress on Evolutionary Computation*, 2012, pp. 1–8.

[8] T. Glasmachers, "A fast incremental bsp tree archive for non-dominated points," in *9th International Conference on Evolutionary Multi-Criterion Optimization - Volume 10173*, ser. EMO 2017. Berlin, Heidelberg: Springer-Verlag, 2017, p. 252–266. [Online]. Available: https://doi.org/10.1007/978-3-319-54157-0_18

[9] M. Drozdík, Y. Akimoto, H. Aguirre, and K. Tanaka, "Computational cost reduction of non-dominated sorting using the m-front," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 5, pp. 659–678, 2015.

[10] R. M. Everson, J. E. Fieldsend, and S. Singh, "Full elite sets for multi-objective optimisation," in *Adaptive Computing in Design and Manufacture V*, I. C. Parmee, Ed. London: Springer London, 2002, pp. 343–354.

[11] S. Mostaghim and J. Teich, *Quad-trees: A Data Structure for Storing Pareto Sets in Multiobjective Evolutionary Algorithms with Elitism*. London: Springer London, 2005, pp. 81–104. [Online]. Available: https://doi.org/10.1007/1-84628-137-7_5

[12] Miqing Li, Jinhua Zheng, and Guixia Xiao, "An efficient mufti-objective evolutionary algorithm based on minimum spanning tree," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 2008, pp. 617–624.

[13] O. Schütze, "A new data structure for the nondominance problem in multi-objective optimization," in *Evolutionary Multi-Criterion Optimization*, C. M. Fonseca, P. J. Fleming, E. Zitzler, L. Thiele, and K. Deb, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 509–518.

[14] A. Jaszkiewicz and T. Lust, "Nd-tree-based update: A fast algorithm for the dynamic nondominance problem," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 5, pp. 778–791, 2018.

[15] M. Jiang, W. Hu, L. Qiu, M. Shi, and K. C. Tan, "Solving dynamic multi-objective optimization problems via support vector machine," in *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*, 2018, pp. 819–824.

[16] C. Cruz, J. R. González, and D. A. Pelta, "Optimization in dynamic environments: A survey on problems, methods and measures," *Soft Comput.*, vol. 15, no. 7, p. 1427–1448, Jul. 2011. [Online]. Available: https://doi.org/10.1007/s00500-010-0681-0

[17] C. Shi, Z. Yan, K. Lü, Z. Shi, and B. Wang, "A dominance tree and its application in evolutionary multi-objective optimization," *Information Sciences*, vol. 179, no. 20, pp. 3540 – 3560, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025509002916

[18] D. Hadka, "Moea framework: a free and open source java framework for multiobjective optimization," 2012.

[19] S. Balaji and M. S. Murugaiyan, in *WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC*, 2012, pp. 26–30.