UNIVERSITY OF EXETER

Computer Science Department

M.Sci. Computer Science
Computer Science Department

Genetic algorithms and distributed optimisation

Candidate

Thomas Bird

Student ID 700046273

Supervisor

Guoqiang Zhang

University of Exeter

Co-supervisor

Academic Year
2023/2024

Abstract

This paper proposes the combination of machine learning and distributed optimisation techniques. This is accomplished through the combination of genetic algorithms with the primal decomposition method. The genetic algorithm is used as a substitution for numerical optimisation techniques to produce a modified version of the primal decomposition method. The modified method is then used to optimise a series of networks with convex problems, and the optimisation is compared to that of the traditional primal decomposition method. There is also usage of the machine learning technique to optimise a local function deemed untenable for numerical techniques.

	Yes	No
I certify that all material in this dissertation which is not my own work has been identified.	☑	☐
I give the permission to the Department of Computer Science of the University of Exeter to include this manuscript in the institutional repository, exclusively for academic purposes.	☑	☐

Contents

List of Figures

1 Introduction

1.1 Project Motivation

1.1.1 Graphs

In the world we live, networks have become more and more prominent, both in mathematical theory, and in practical applications. Physically, sensor networks [3], internet connections in the form of peer to peer networks [15], or power networks [22]. Theoretically, they can be representations of many things, provided there is are pre-defined points that carry information, and that they are connected to other points. Simply put, anything with connections to other things has the possibility of being represented by a graph, or a network of nodes. For example, cities are connected by roads. The cities may be represented as nodes, and the roads may be represented by connections between each node. With a graphical representation of a network, values and constraints may be recorded on each connection and node. Following the city example, each node could contain variables relating to the population, whilst each connection could contain variables relating to the distance of the road between the two cities. These variables could be based on different values for the problem that is being considered, and more complex amounts and relationships may be represented by different mathematical functions. These mathematical functions may be constrained as a result of the problem. Continuing with the city example, a variable attached to a node may represent the population, whilst the edges to other nodes may represent the number of commuters. In this scenario, a constraint may be placed on the connection between two cities, which would prevent the number of possible commuters exceeding the population of both cities.
The modelling of this situation would often create a graph that appears like the following:

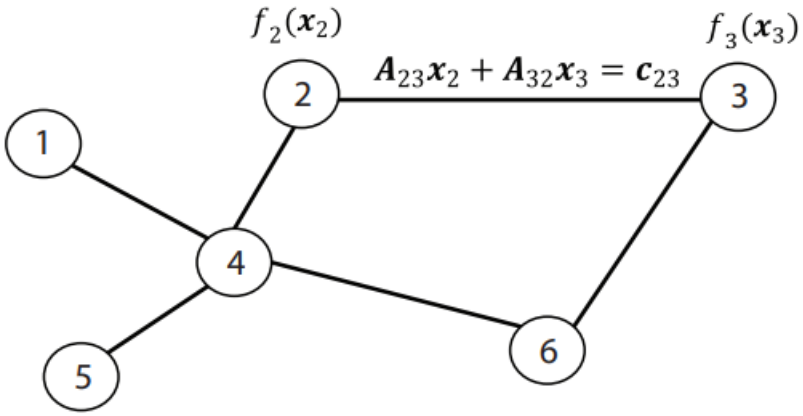

Figure 1.1: Example of a graph with constraints, and the sum of functions on each node [23].

Due to their versatility, many different problems can be represented through the use of graphs, allowing a wide variety of potential uses. As a result of the numerical nature of graphs, and the wide field of graph theory, the potential for optimisation of networks is both a viable and valid possibility. As such, distributed optimisation has become a well researched field, focusing on the optimisation of numerical functions over a network.

1.1.2 Distributed Optimisation method review

Although covered in the literature review, a review of the methods of distributed optimisation will be covered here to provide context for the project. When optimising functions over a network, many different approaches have been theorised and tested. As previously covered in my literature review, several different approaches exist, making use of different fundamentals to accomplish the goal of network optimisation. These will be discussed here to provide a foundation for the general ideas and thought processes that go into the creation of different distributed optimisation techniques, providing background context to the processes involved.

When looking at distributed optimisation, we must first consider the mathematical objective function to optimise. In the context of a graph, there will be three key components that make up the challenge of optimisation. Firstly, there will be the functions of each node, which may be referred to as a cost function. These are mathematical functions attached to each node that are be applied to the second component, the decision variables of a node. Sets of decision variables are contained on each node, and are often restricted in value by the third and final component, the constraints placed on edges. Often when modelling networks, certain restrictions exist for certain properties, and these restrictions may be based around the connections between parts of the network. When modelling these restrictions, the constraints are often attached to edges to show the limitation between the two variables. All of these components may be assembled into a graph, where nodes have decision variables, that are restricted by constraints, that are transformed into a cost value by the cost function on the same node. This allows the creation of a graph similar to those found in figure 1.

Often, the sum of all cost functions in the network will want to be optimised to be maximised

or reduced, based on the problem. This may be accomplished by finding the optimal values for decision variables across all nodes in the network. This may be represented by the formula:

$$F(x) = \sum_{i \in V} f_i(x_i) \quad \text{subject to } A_{ij}x_i + A_{ji}x_j = c_{ij} \quad \forall (i,j) \in E, \tag{1.1}$$

Before analysing individual techniques, a distinction should be made between two types of algorithm, relating to the order each step is taken in a given algorithm. When optimising over a network, multiple steps are often taken, ensuring an optimisation of the network, where nodes often update their own values multiple times according to changes within the network. Frequently, for simulation purposes, the amount of steps is bounded by a number, and each step is referred to as an iteration. The number of iterations a node goes through is usually the same, however the timing each node proceeds to the next iteration is based on the two distinct types of algorithm, referred to as "synchronous" and "asynchronous". A synchronous algorithm is an algorithm where each node receives an update to its variables, before waiting until each other node has been updated at least once, after which it proceeds to the next iteration. An asynchronous algorithm differs from synchronous algorithms in the fact a node can proceed to the next iteration without confirmation that other nodes within the network have finished processing their current iterations.

Another consideration to make when comparing distributed optimisation algorithms relates to signal loss. Distributed optimisation has often found usage in real world scenarios when mapping out networks that transfer radio signals [24]. However, when operating with these types of signals, messages passed between nodes can become disrupted, either reducing the amount of information they carry, or completely preventing the message from reaching their destination. In these cases, the performance of an algorithm can be quite heavily impacted, as such, considerations can be made about the viability of a specific algorithm for a task based on its resilience to signal loss.

Now that the distinction between the types of algorithm has been made, there will be a discussion of past distributed optimisation methods, with their advantages and disadvantages discussed, to help provide information relating to choices within the project.

Gossip based methods and path averaging

The first methods to look into are the uniform and weighted gossip methods respectively. These two methods have been grouped together due to the similarities between the two techniques, and can be seen as extensions of the randomized gossip scheme [4]. Both techniques are based around the goal of averaging the values between all nodes into a similar value. To accomplish this goal, each node at each iteration will choose a random neighbour, and transfer a message to it, with the contents of the message being dependent upon the algorithm. These messages will contain information relating to the desired value of the nodes variables, or the optimisation of the cost function. The way each node selects a destination node to transfer to is where the algorithms differ. Within uniform gossip, the destination node selected is ran-

domly chosen [4] with a uniform probability based on its neighbours. Weighted gossip, on the other hand, begins with each node constructing a "diffusion matrix", which features a series of weights for each node that influence whether a specific node is chosen randomly, with higher weighted nodes having a greater chance of being selected [1]. Weighted gossip has several advantages over randomised gossip algorithms as it allows emphasis on more useful nodes within the network, ensuring faster convergence across the network. Randomised gossip does not possess this advantage, however, they are easier to implement as weights for the diffusion matrix do not need to be considered.

Although these algorithms have been proven to be beneficial at achieving goals in the context of distributed averaging, they often lack the capabilities of handling more complicated optimisation problems, especially when constraints are included.

Examples of other algorithms may be found within the prior literature review, as well as among some citations. However, for the sake of brevity, two key methods will be discussed to provide background context. These are the alternating direction method of multipliers, and the primal dual-method of multipliers, or ADMM and PDMM respectively.

ADMM and PDMM

When considering the optimisation of more complex problems, ADMM and PDMM both prove to be effective at the task of optimising convex functions across networks.

These methods are covered with more depth within the literature review, and the specifics implementations of each method will be discussed during the design section of this report. However, to briefly cover their functionality, they optimise convex cost functions subject to constraints across edges within the network through a series of mathematical steps. Both methods model the constraints placed upon a given node through the use of the Lagrangian functions [21] [5] [23]. Through the use of the Lagrangian functions, the functions are transformed from a constrained form into an unconstrained form, allowing the use of traditional optimisation techniques to calculate optimal decision variables.

The optimisation of unconstrained convex functions is also a widely researched topic with a growing variety of solutions, and a broad range of applications. Two of these techniques will now be covered to provide context for the design of the project, with additional information being available within the literature review.

Distributed Primal Decomposition

A recently developed method based around similar concepts to PDMM is the distributed primal decomposition [8]. Through the use of Lagrangian functions and dual variables, distributed primal decomposition allows the optimisation of constrained problems as unconstrained, whilst also preserving the constraints within the network. The algorithm itself has multiple benefits to it, such as computation being bound to each node, whilst also having performance comparable to other state of the art algorithms [8].

1.1.3 Local Optimisation Techniques

Genetic Algorithms and Evolutionary Strategies

Genetic algorithms and evolutionary strategies are two techniques used for the optimisation of complex functions to produce optimal outputs. Both techniques operate on very similar principles, focused on generating populations of viable solutions for the problem, and evaluating each solution based on how effectively they accomplish the objective function. The population members are then selected and used to make the next generation based on a collection of functions. Through the iterative process of making generations from previous ones, the solution space is explored, with the preferential selection of high scoring individuals for the creation of future generations. Ideally, this will result in an approach towards optimal solutions to the objective function [17] [2].

Bayesian Optimisation

Bayesian optimisation is another option for optimisation of complex functions, based around the sampling of an objective function, and constructing a surrogate function that represents an approximation of the objective function. Quite often, the surrogate function will be relatively less computationally expensive to evaluate compared to the original objective function, allowing techniques that require multiple samples of an objective function to be ran comparatively faster using the surrogate function. Additionally, the surrogate function has the advantage of including probabilities functions that can describe the accuracy of the surrogate function at a given point [12].

1.1.4 Project Motivation

With these methods of distributed optimisation covered, along with the necessity of graph based optimisation in the modern world, the development of these techniques can lead to avenues where the applications of improvements are applicable to a wide variety of technological areas.

Additionally, with the development of new local optimisation techniques, such as genetic algorithms and Bayesian optimisation, the ability to consider and optimise substantially more complex equations has become increasingly viable.

This project aims to bridge these two areas, to show that the use of different local optimisation techniques can be combined with distributed optimisation techniques to begin by providing a viable alternative to the mathematical optimisation techniques used at a nodes base level, whilst also showing the potential for distributed optimisation of substantially more complex problems that normally would be incapable of optimisation through numeric techniques.

1.2 Project Scope

Now that the motivation for the project has been covered, the projects objectives and scope will now be covered to provide direction for how to best realise the motivation.

1.2.1 Primary Objectives

The primary objectives of this project are as follows:

1. Prepare optimisation techniques for local problems
2. Implement a modified version of a distributed optimisation algorithm
3. Create a method of testing the modified algorithm
4. Record performance statistics from the testing of the modified algorithm.
5. Ensure that the modified algorithm is capable of optimising basic problems

With the completion of these objectives, the project will ideally have produced and showcased an alternative method to local optimisation to distributed optimisation. Provided this is accomplished, the potential for investigation into further techniques, and for use with more complex problems, can be pursued, by either this project, or future papers, which leads to the secondary objectives.

1.2.2 Secondary Objectives

With the completion of the primary objectives, the viability of further research would be open to investigation. The secondary objectives focus on this in two key aspects, by evaluating the performance of the modified algorithm in comparison to other algorithms, and attempting to optimise complex objective functions across nodes that are difficult to analyse using the standard numerical methods. The first secondary objective will be based around performance analysis, and is dependent on how effectively the modified algorithm can match the performance of a non-modified algorithm. Ideally, the modified algorithm will be able to reduce the sum of the objective function across all nodes in a graph to a similar degree of a non modified algorithm. If this objective is achieved, then the modified algorithm could be used as a substitute to previous methods, allowing more in depth study of the algorithm as a whole. The other secondary objective is dedicated to providing options for distribution optimisation of more complex problems that may previously be seen as untenable. Assuming the modified algorithm is capable of approaching solutions in tasks where numerical techniques may be unusable, then it potentially allows further study of distributed optimisation for previously unfeasible tasks.

1.3 Specification

1.3.1 Background

As previously discussed in both this document and the literature review, there are a wide variety of techniques that can be used to optimise a distributed network, with many different works focusing on algorithms and their performance. Additionally, there has been a great deal of work focusing on distributed evolutionary strategies, however, like previously mentioned in the literature review, these often use several different layouts to accomplish the goal of optimisation of a single objective function not attached which are not attached to any node [19] [14]. As such, these are more focused on the use of large amounts of distributed computational resources to accomplish iterations of single runs of evolutionary strategies, as opposed to focusing on the optimisation of a network itself. There are also papers that discuss the use of distributed optimisation with objective functions that are non-convex [20] [16], however these papers focus solely on the accomplishment of the distributed optimisation task, and do not discuss the use case for machine learning techniques for local optimisation.

1.3.2 Evaluation

To conclude our introduction, I will consider the objectives, and how each objective may be evaluated. The system should be able to complete all of the following tasks:

1. **Local optimisation technique** - Machine learning technique that can optimise an objective function
2. **Modified technique implementation** - Code for running a distributed optimisation algorithm with the machine learning technique
3. **Testing methods** - Create tests that can be run with the modified algorithm
4. **Record performance statistics** - Extract information from the tests that allow performance analysis
5. **Basic problem solving** - Run the tests with notably similar results to previously established techniques

Provided the project completes these tasks, then the study of the modified algorithm in comparison to other algorithms will be possible, with the possibility of introducing additional functionality that will allow the completion of secondary objectives.

2 Implementation

2.1 Design

Firstly, this program will be developed using Python to use various libraries to accomplish the objectives. The initial design of the program is focused on completing the primary objectives. As such, there are four key components that must be designed and implemented. These consist of:

1. Graph with functions and constraints
2. Method of local optimisation
3. Distributed Optimisation method
4. Test suite

2.1.1 Graph with functions and constraints

To create graphs with both functions and constraints available for testing of distributed optimisation techniques, the python library Disropt [11] will be used. The library features a wide variety of classes and functions that allow creation of graphs with objective functions and constraints. Additionally, the library comes with several distributed optimisation methods. These will be covered later in this report.
The library itself represents nodes within the graph via the use of singular programmed agents. Each agent is its own separate process that runs simultaneously to all other agents via the use of the MPI for Python library[9]. The MPI for Python library allows the use of the message passing interface with the Python language, allowing the multiple processes to communicate, effectively simulating the communication of nodes in a network, allowing them to each run in parallel. This does, however, have several effects on the project.
Firstly, the number of nodes within a graph will be decided by the number of processes created.

This does mean having larger numbers of nodes will likely result in performance loss. Because of this, the assumption has been made that the processes will be a reasonable representation of the individual processing power of a given node.
Secondly, as each node will be running independently of other nodes, they are likely to finish their calculations for a given iteration at different times. Due to this, the transfer of variables between nodes will likely take place at differing times. This results in similar behaviour to the previously discussed asynchronous schemes. As such, the assumption has been made that the nature of the irregular transfer intervals is a suitable representation of an asynchronous scheme, and that the level of irregularity produced will not be detrimental to the accuracy of the project.

2.1.2 Genetic Algorithm Implementation

For the implementation of genetic algorithms within the project, the python library PyGAD [13] will be used. The library itself features a wide range of options that allow changing of variables and hyper-parameters when constructing the genetic algorithms. It should be noted that these will be used for the optimisation of the Lagrangian functions on a given node. Details on how this will accomplished will be discussed later in the project.

2.1.3 Distributed optimisation method with genetic algorithm

Contrary to the literature review, the distributed primal decomposition method has been chosen due to its advantage over other algorithms [8], and its relative ease of use. Additionally, the algorithm has the potential to be expanded for use with mixed integer linear programming [7] [6]. However, due to both this algorithm and PDMM sharing fundamental roots, exploration of changes in one may suggest similar changes in the other.
The algorithm itself will be modified through the addition of the genetic algorithm into its optimisation structure. An implementation of the method to this process is included in the Disropt library, in the distributed primal decomposition algorithm.
To provide insight into this process, the distributed primal decomposition pseudo-code will be displayed, with explanations on how this is used and completed in the project. To begin with, we start by looking at the following pseudo-code [8].

Algorithm 1 Distributed Primal Decomposition

Initialization: y_i^0 such that $\sum_{i=1}^{N} y_i^0 = 0$

for $t = 0, 1, 2, \ldots$ **do**

Compute (x_i^t, ρ_i^t), μ_i^t as a primal-dual solution of

$$\begin{aligned} &\min_{x_i, \rho_i} f_i(x_i) + M\rho_i \\ &\text{subj. to } \mu_i : g_i(x_i) \leq y_i^t + \rho_i 1 \\ &x_i \in X_i, \rho_i \geq 0 \end{aligned} \tag{2.1}$$

Gather μ_j^t from $j \in N_i^t$ and update

$$y_i^{t+1} = y_i^t + \alpha^t \sum_{j \in N_i^t} (\mu_i^t - \mu_j^t) \tag{2.2}$$

end for=0

Fundamentally, this pseudo-code does not change when considering the creation of the alternative method. Instead, the main changes to make are during the minimisation stage within the first section. Normally, this would be done using a numerical technique, but for the project, a genetic algorithm will instead be used. From the figure, it should be noted that the Disropt library makes use of several classes to run the distributed optimisation algorithms. These consist of an agent, problem, and algorithm class respectively. The agent class is primarily used for the simulation of the graph, by storing variables and transferring messages to other connected nodes. The problem class is used for storing information relating to the objective functions of each agent, including constraints placed upon the objective function. Finally, the algorithm class contains the functional processes a node will go through during optimisation. To achieve the goal of the project, the normal Disropt algorithm class will instead be substituted for the modified version, with the associated changes to the solution process. During the optimisation process for Primal Decomposition, a smaller sub-problem is created, so that it may be evaluated through the use of CVXPY [10], a convex optimisation library that makes use of mathematical techniques for problem solving. As this is where the key difference between the normal and modified algorithm occurs, a substitute problem class will also be created that instead uses genetic algorithms for optimisation as opposed to CVXPY.

Genetic Algorithm within Primal Decomposition

Primarily, the new population algorithm will be where the most challenge is found. CVXPY handles optimisation completely, including the management of dual variables produced from the Lagrangian function. However, due to the nature of PyGAD, the Lagrangian function must be constructed within the problem to accomplish its goal. Within the context of Primal Decomposition, the Lagrangian is the objective function to optimise. However, the function itself

must be constructed. Within the project, this will be handled by the modified problem class. The modified problem will need to calculate optimal values for both x_i and ρ_i when solving the Lagrangian. As PyGAD treats each population member as a list of genes. The list will consist of two parts, an x section, and a ρ, section respectively. Each gene within each section will refer to a single value that functions as an input for the Lagrangian either as a depending upon which section it is in.
Additionally, PyGAD requires an initialisation range when creating new populations, or mutating certain genes. Because of this, two ranges of genes will need to be created for the ρ and x values respectively. Two approaches have been developed for this task.
The first, is arbitrarily deciding two values to select for the initialisation ranges of ρ and x. As this would entirely be up to the developer, and may have no basis on the actual problem, this initialisation process will be referred to as the uninformed genetic algorithm process.
The second approach involves deciding the range for x based on the first constraint, by evaluating the function provided with it, essentially representing $g_i(0)$ for that constraint. The value of this result will be used as the upper and lower bound by changing the sign of the value respectively. This technique, however, is founded on the prior knowledge that the local objective functions used within tests will include a set of box constraints. Due to this, the technique will be used for the standard genetic algorithm tests. It is worth acknowledging, however, that this technique benefits from prior knowledge, and as such, may require consideration in future study. Every generation after the first for genetic algorithm optimisation uses the current x values at that iteration.

2.1.4 Testing Suite

Finally, all of these components will be joined together within a testing suite. Due to the nature of MPI for Python, the test suite will be called from the command line along with a sequence of arguments that will control the number of processes assigned to a test (affecting the number of nodes within a simulated graph), along with other parameters for varied tests. The option for testing other distributed optimisation techniques will also be available, allowing comparisons to be made.
The tests will save information according to the total cost function across all nodes within a graph at each of their respective iterations. This will allow a visual representation of the how effectively each algorithm accomplishes the distributed optimisation task.
This results in the following layout for the program.

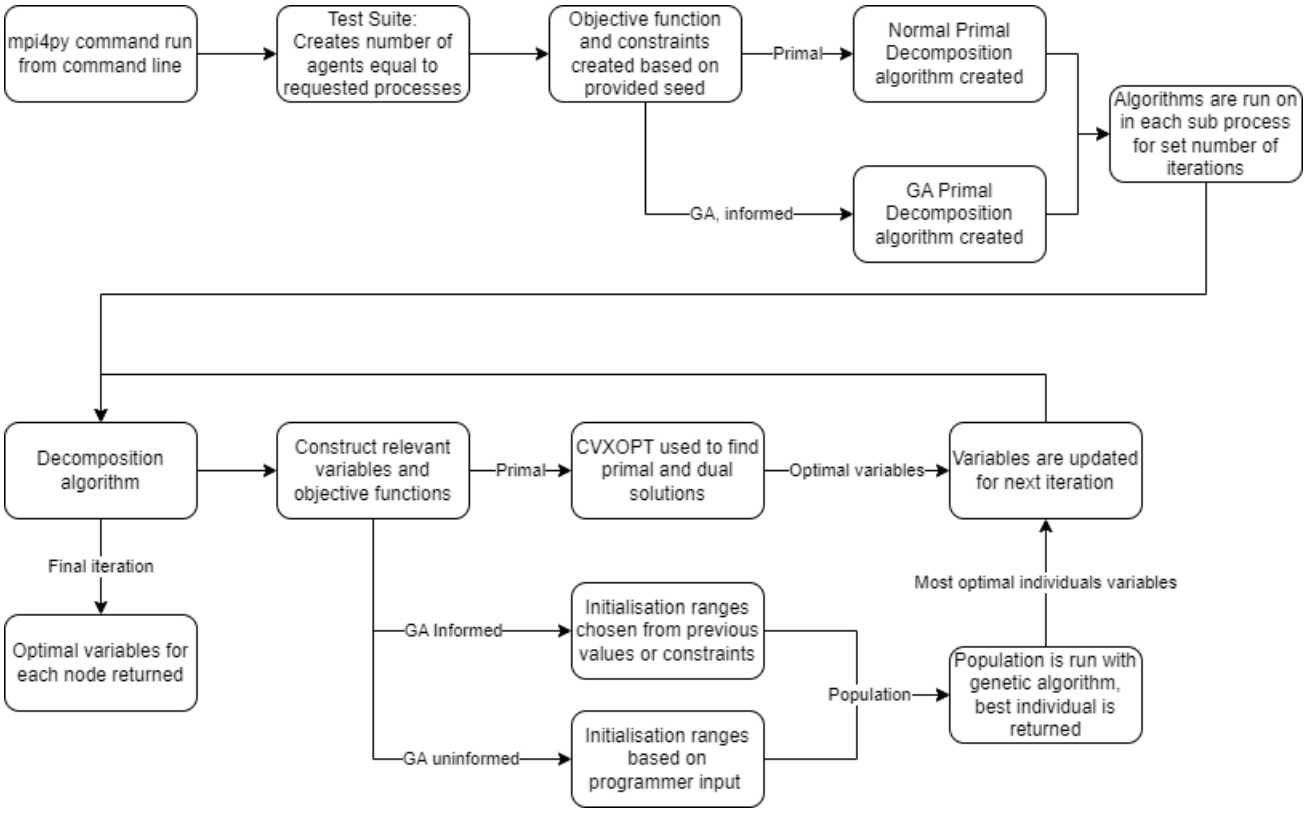

Figure 2.1: Visual representation of classes and communications within the project

3

Testing and Results

3.1 Introduction

As testing begins, we must first consider the initial variables that will be decided. The test suite currently has five arguments available that influence the creation of the problems to optimise. The first variable relates to the number of nodes within the network graph. The second variable relates to the random seed used in the formulation of the objective function. The third variable relates to the number of constraints a given function has. The fourth variable relates to the dimensions of the problem, affecting the number of local variables for a problem. The fifth variable relates to the method that will be chosen for a given optimisation task.
With these variables decided, these can be used to create a wide range of problems that can be used to evaluate the effectiveness of a given problem.

3.1.1 Initial Tests

To begin with, a basic test was run with each algorithm to establish the functionality of the modified algorithm. Initially, this was done using a graph with 2 nodes, 2 variables and 1 constraint. This was done mainly to establish the functionality of the project, as opposed to a direct performance analysis. During these test, two observations were made.
Firstly, due to the asynchronous nature of MPI, multiple runs using the same random seeds were often different, as the sub processes would run at different speeds and influence each others algorithms. Additionally, after the running of a similar problem multiple times on a machine, the functions began to have faster convergence rates, likely due to the process. To reiterate, this is likely a side effect of running MPI on a single machine, as opposed to other ways of simulation.
Secondly, the initialisation range heavily affected performance, likely due to the stochastic nature. During initial tests, the uninformed genetic algorithm method would constantly produce rapidly varying levels of fitness with no signs of convergence. When a random seed was in-

troduced to ensure the genetic algorithm followed a more predictable nature, the uninformed process did not increase or decrease in its performance, likely due to the initialisation process not accounting for previous changes within the x values, despite the fitness function changing between iterations.
From the initial tests, the following graphs was produced:

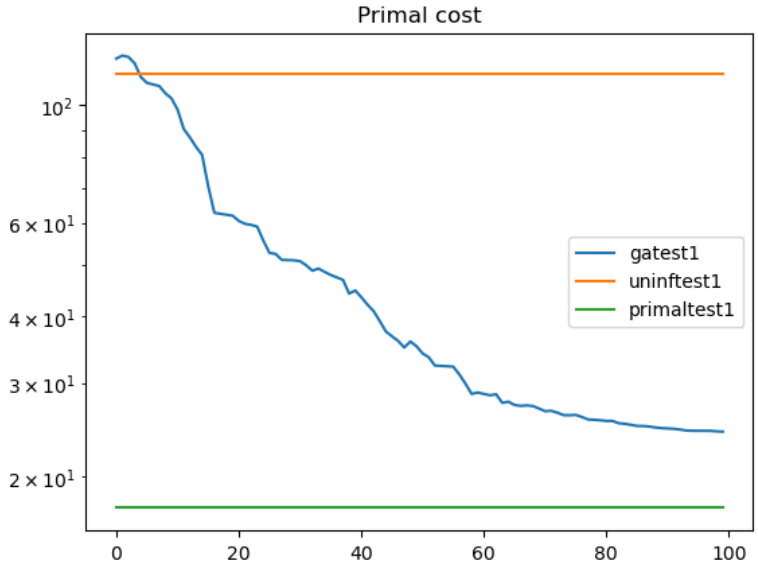

Figure 3.1: Performance characteristics of the initial tests. The x axis refers to the number of iterations t of the algorithm, whilst the y axis refers to the total primal cost of all nodes within the graph

The labels of the lines represent the tests, with uninf1 referring to the test of the uninformed method of initialisation with the modified distributed primal decomposition. The ga1 refers to the modified version of the same algorithm using the informed initialisation technique. Finally, primaltest1 refers to the test of the unmodified algorithm.
Although imperceptible on this graph, the unmodified variant did converge towards a minimum value. However, this was after several repeated runs on the same machine, and as such, is likely an unintended side effect of MPI.
It is also worth mentioning that the axes used on this graph will be repeated for all other graphs until mentioned otherwise.
From this initial test, it can be observed that the modified algorithm can converge towards a minimum that the unmodified version of the algorithm is capable of achieving. However, it is noticeably slower in convergence at first glance, and it can be seen visually that the overall primal cost begins at an exceptionally high value in comparison to the standard method.
This initial set of results suggests several things about the project itself. Most notably, that the modified algorithm is capable of optimising, provided the initialisation of values for the genetic algorithm are sensibly influenced by previous runs of the algorithm. Additionally, prior knowledge of the problem to optimise can be useful.
Due to the ambiguity of these results, a set of new tests will be run with more complex algo-

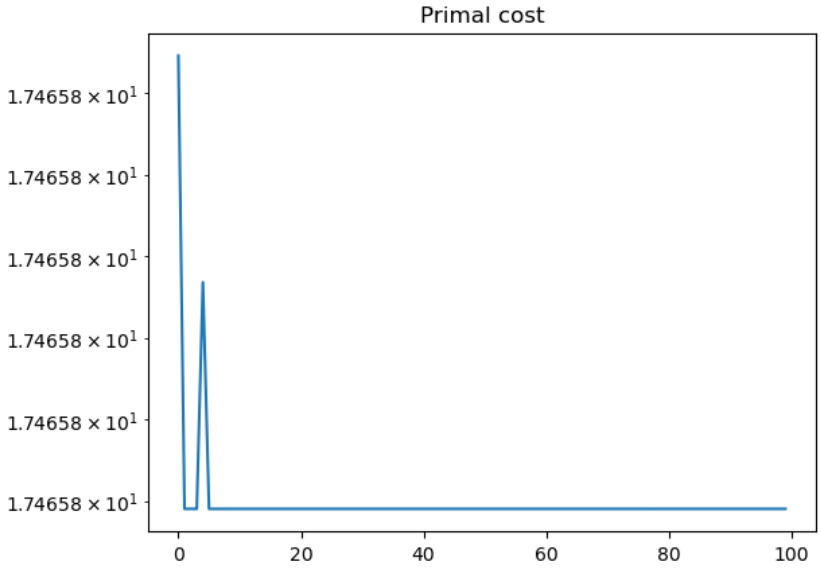

Figure 3.2: Performance characteristics of the unmodified distributed primal decomposition method after several runs on the same machine

rithms to visualise how these algorithms perform in comparison to each other.

3.1.2 Additional nodes

After running a similar simulation, this time, with 4 nodes as opposed to 2, whilst keeping the number of constraints and dimensions constant, the following output was produced:

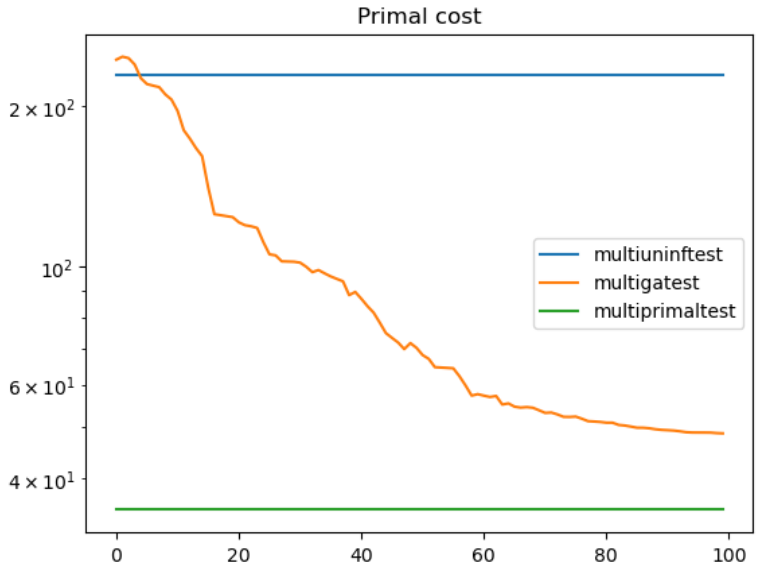

Figure 3.3: Performance characteristics of the test featuring additional nodes within the graph

The results of this graph is very reminiscent of the previous graph. For the sake of brevity, the unmodified algorithm featured a similar output to the previous one, wherein there was convergence, at an exceptionally small magnitude, in an exceptionally small number of iterations.

3.1.3 Increased objective function complexity

After running a simulation, with 2 nodes, but with increases to the number of both the constraints and dimensions to 3, the following output was produced:

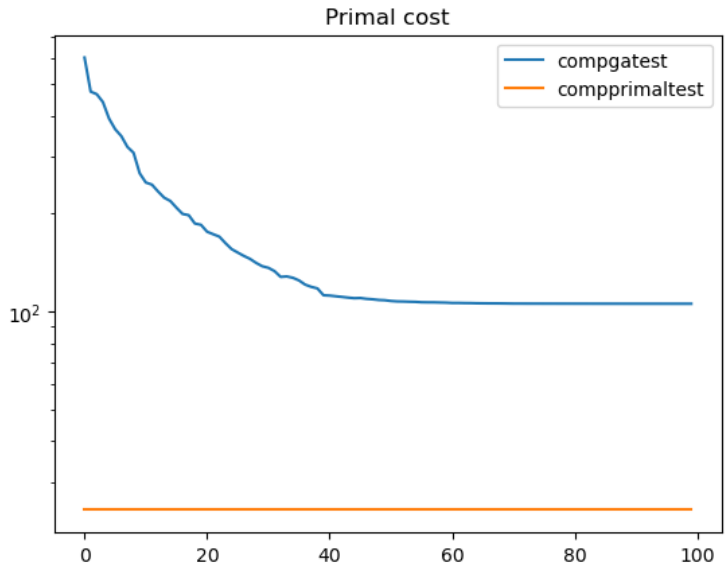

Figure 3.4: Performance characteristics of the tests featuring additional constraints and decision variables within the graph

To begin with, the uninformed method was excluded from the graph, as it resulted in similar results to the prior tests. This is likely due to a programming error. That aside, the graph shows that the modified technique is capable of handling somewhat more complex problems. However, the plateauing of the graph towards the end of the iterations suggests that the modified algorithm may have been converging towards a local minima.

3.1.4 Complexity and node count

The following output was produced after running a test using both the increased constraint counts and dimensions to 3, and the number of nodes within the network to 4.
A similar pattern to before may be observed in this graph. Once again, the modified algorithm begins at a high value and slowly converges to what appears to be a local optima, whilst the convex optimisation technique begins already at a substantially better optimal value, and does not change in any comparable way.

3.1.5 Infeasible

During testing, a function was constructed that could not be solved using the available CVXOPT techniques for the given task, preventing an output from the algorithm. Although this is likely a choice by the developers of the respective libraries, it did present an opportunity for testing the modified algorithm in a setting considered unfeasible for numerical techniques.

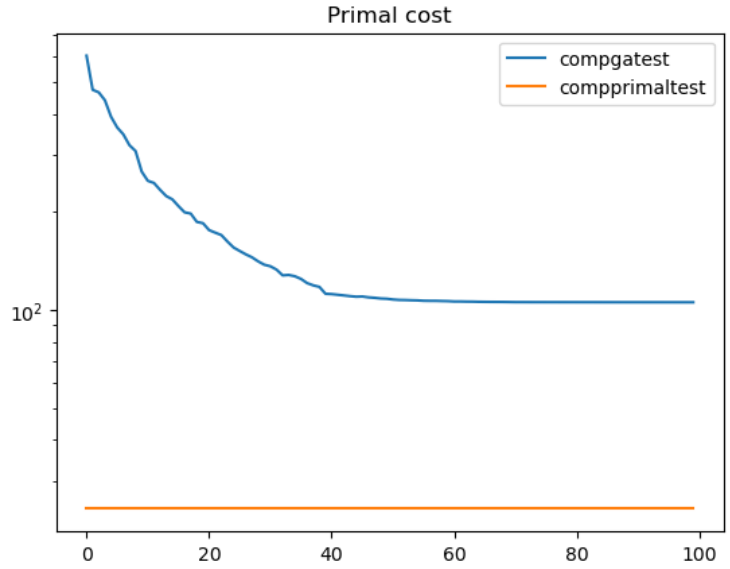

Figure 3.5: Performance characteristics of the tests featuring additional constraints, nodes and decision variables within the graph

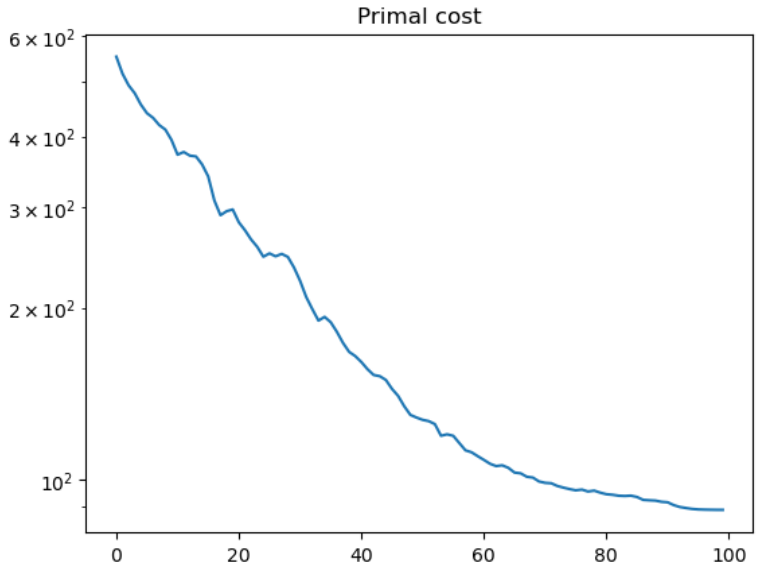

Figure 3.6: Performance characteristics of the modified algorithm with a problem considered numerically infeasible by CVXOPT

In this scenario, the algorithm performed in the following way:
Although there is little comparison to be made in terms of performance, and this is only one example of optimisation with a function that could not be completely solved by the techniques provided by CVXOPT, it does suggest that there is the feasibility for exploration of search spaces that numerical techniques could not feasibly solve.

4 Conclusion

4.1 Results Discussion

From the series of tests, we can begin to draw conclusions about the viability and performance of the modified scheme in comparison to previously developed techniques. Although in all test cases, the non-modified technique featured lower global primal costs that were achieved in substantially fewer iterations than the modified technique, the fact that the modified technique seemed to approach local optima in most cases suggests that the potential for use of genetic algorithms for local optimisation in the context of distributed optimisation. Additionally, in the case where traditional optimisation techniques were unusable, the modified technique proved to at least output partially optimised solutions, which suggests that the option for using the modified technique may be prove to be useful for more complex optimisation tasks.
Overall, the results suggest that the primary objectives have been at least partially achieved, meaning the project serves as a proof of concept for the introduction of more machine learning techniques into local optimisation for distributed optimisation, provided consideration of the formulation of techniques is taken. This is primarily seen with the effectiveness of the informed technique compared to the uninformed technique, although this may just be as a result of programming errors.

4.2 Flaws

There are a number of flaws within this paper and project that both call into account the validity of the claims made, and leave gaps within the research of this technique.
The first major flaw is the lack of depth to each test. Due to the time constraints, number of potential variables, and small amount of information recorded, the tests do not suggest many different factors outside of differences in performance between schemes. Although the depth of tests was not the main focus of the project, additional information provided could help inform

future study within the area, as well as provide potential guidance if the scheme was going to be used for a specific task.
The second largest flaw is the lack of both variety and numerical information in regards to the performance metrics. Continuing from the previous point, performance is only vaguely covered from a visual perspective, as opposed to an explicitly numerical comparison of the results. Additionally, the consideration and recording of more metrics could also have been insightful. Some examples include run time, gradients of optimisation, and evaluation counts. Further potential metrics can be found in [8] [18].
The project also suffers from a lack of deterministic nature. Although the option of using the same seed to form functions, as well as each genetic algorithm run being seeded allows consistent tests on those fronts, the lack of a symmetrical, deterministic method means the results obtained both from running the program and from the report are difficult to replicate.
There is also the key issue of the lack of information relating to both the objective functions, and the output variables. Although the input functions and output variables are obtainable from within the records generated by the program, these are not included in the report itself, potentially leading to missing information that could be beneficial to future research.
Finally, the potential errors within the program, and the lack of algorithmic analysis, calls into question the validity of the claims made from the results, and the results themselves. The unexplained nature of the behaviour of the uninformed method and the rapidly optimising and plateauing actions of the standard technique also call into account the accuracy of these implementations, and indirectly, the legitimacy of the rest of the paper as well.

4.3 Future Directions

The future directions for this project and report would come from fixing the code (assuming it is not functioning as intended), amending the flaws and extending the range of possible distributed optimisation tests.
If the code was fixed there would be more credibility for the claims made in this report, and more tests could be performed. This would be greatly beneficial as it could potentially offer new insights into the data that may have been missed due to coding errors.
Whilst considering new insights, managing the flaws within the project would be greatly beneficial, as it would allow consideration of other statistics relating to an algorithms effectiveness. With a wider range of options for evaluating algorithms, the potential for many more insights into the algorithms used.
Leading on from this, the number of algorithms studied can be increased, both for the distributed optimisation component, and for the local optimisation component. There are many other machine learning techniques available, including the previously mentioned evolutionary strategies, and Bayesian optimisation. These could be combined with other techniques that make use of optimisation, such as the previously covered PDMM. By doing this, we may see certain methodologies perform better than others when combined with machine learning techniques.

Overall, I believe the paper has given a surface level insight into how a machine learning technique could be integrated into a distributed optimisation algorithm. The primary objectives focused on this as a goal, aiming to show that the possibility for the combination for these two techniques may be a viable option. Additionally, despite there being only a single test case within the paper, the potential for the use of machine learning techniques when numerical methods prove ineffective for optimisation is showcased. Due to this, a portion of the secondary objectives can be considered completed.
Assuming their is enough information within the project to inspire and support future endeavours that make use of the combination of machine learning techniques alongside distributed optimisation techniques, then this project can be seen as a success, albeit a minor one. The primary goal was to showcase a proof of concept, and I believe that this project has the potential to inspire other works to further research and develop these techniques.

References

[1] Florence Bénézit et al. "Weighted Gossip: Distributed Averaging using non-doubly stochastic matrices". In: *2010 IEEE International Symposium on Information Theory*. 2010, pp. 1753–1757. DOI: 10.1109/ISIT.2010.5513273.

[2] Hans-Georg Beyer and Hans-Paul Schwefel. "Evolution strategies–a comprehensive introduction". In: *Natural computing* 1 (2002), pp. 3–52.

[3] S. Boyd et al. "Randomized gossip algorithms". In: *IEEE Transactions on Information Theory* 52.6 (2006), pp. 2508–2530. DOI: 10.1109/TIT.2006.874516.

[4] S. Boyd et al. "Randomized gossip algorithms". In: *IEEE Transactions on Information Theory* 52.6 (2006), pp. 2508–2530. DOI: 10.1109/TIT.2006.874516.

[5] Stephen Boyd et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: *Foundations and Trends® in Machine learning* 3.1 (2011), pp. 1–122.

[6] Andrea Camisa, Ivano Notarnicola, and Giuseppe Notarstefano. "A primal decomposition method with suboptimality bounds for distributed mixed-integer linear programming". In: *2018 IEEE Conference on Decision and Control (CDC)*. IEEE. 2018, pp. 3391–3396.

[7] Andrea Camisa, Ivano Notarnicola, and Giuseppe Notarstefano. "Distributed primal decomposition for large-scale MILPs". In: *IEEE Transactions on Automatic Control* 67.1 (2021), pp. 413–420.

[8] Andrea Camisa et al. "Distributed constraint-coupled optimization via primal decomposition over random time-varying graphs". In: *Automatica* 131 (2021), p. 109739.

[9] Lisandro Dalcín, Rodrigo Paz, and Mario Storti. "MPI for Python". In: *Journal of Parallel and Distributed Computing* 65.9 (2005), pp. 1108–1115. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2005.03.010. URL: https://www.sciencedirect.com/science/article/pii/S0743731505000560.

[10] Steven Diamond and Stephen Boyd. "CVXPY: A Python-Embedded Modeling Language for Convex Optimization". In: *Journal of Machine Learning Research* (2016). To appear. URL: https://stanford.edu/~boyd/papers/pdf/cvxpy_paper.pdf.

[11] Francesco Farina et al. "DISROPT: a Python Framework for Distributed Optimization". In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 2666–2671. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2020.12.382. URL: https://www.sciencedirect.com/science/article/pii/S2405896320306686.

[12] Peter I Frazier. "A tutorial on Bayesian optimization". In: *arXiv preprint arXiv:1807.02811* (2018).

[13] Ahmed Fawzy Gad. "Pygad: An intuitive genetic algorithm python library". In: *Multimedia Tools and Applications* (2023), pp. 1–14.

[14] Yue-Jiao Gong et al. "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art". In: *Applied Soft Computing* 34 (2015), pp. 286–300. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2015.04.061. URL: https://www.sciencedirect.com/science/article/pii/S1568494615002987.

[15] D. Kempe, A. Dobra, and J. Gehrke. "Gossip-based computation of aggregate information". In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* 2003, pp. 482–491. DOI: 10.1109/SFCS.2003.1238221.

[16] Ion Matei and John Baras. *A non-heuristic distributed algorithm for non-convex constrained optimization*. Digital Repository at the University of Maryland, 2013.

[17] Seyedali Mirjalili and Seyedali Mirjalili. "Genetic algorithm". In: *Evolutionary Algorithms and Neural Networks: Theory and Applications* (2019), pp. 43–55.

[18] Angelia Nedic, Alex Olshevsky, and Wei Shi. "Achieving geometric convergence for distributed optimization over time-varying graphs". In: *SIAM Journal on Optimization* 27.4 (2017), pp. 2597–2633.

[19] Lu Sun et al. "Large scale flexible scheduling optimization by a distributed evolutionary algorithm". In: *Computers & Industrial Engineering* 128 (2019), pp. 894–904.

[20] Tatiana Tatarenko and Behrouz Touri. "Non-convex distributed optimization". In: *IEEE Transactions on Automatic Control* 62.8 (2017), pp. 3744–3757.

[21] Ermin Wei and Asuman Ozdaglar. "Distributed Alternating Direction Method of Multipliers". In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. 2012, pp. 5445–5450. DOI: 10.1109/CDC.2012.6425904.

[22] Tao Yang et al. "A survey of distributed optimization". In: *Annual Reviews in Control* 47 (2019), pp. 278–305.

[23] Guoqiang Zhang and Richard Heusdens. "Distributed optimization using the primal-dual method of multipliers". In: *IEEE Transactions on Signal and Information Processing over Networks* 4.1 (2017), pp. 173–187.

[24] Guoqiang Zhang, Richard Heusdens, and W Bastiaan Kleijn. "Large scale LP decoding with low complexity". In: *IEEE Communications Letters* 17.11 (2013), pp. 2152–2155.