Research infrastructure for PARALLEL GP SYSTEM EXPERIMENTS in Portfolio Optimisation

Name: Ali Adeli

Supervisor: Dr. Christopher D.Clack

Year of Submission: 2006/7

Programme: BSc Computer Science

Acknowledgment

I would like to express my sincere gratitude and appreciation to my supervisor Christopher D. Clack for his invaluable inputs and the passionate assistance provided by him during this academic year. I would also like to thank Ghada Hassan for spending her time helping me understand and develop my project.

Finally, I would like to thank my family for their continuous support throughout my studies and for making this happen.

Abstract

Genetic Programming (GP) is one of the most widely used machine learning techniques, for portfolio optimisation. The breadth of its use goes as far as stock market prediction, advanced mathematics, military applications and the most relevant to this project, financial forecasting. Evaluation of the GP system is very essential. Nevertheless, time constraints in terms of processing are a major factor concerning limitations of the system. For this reason, the aim of this project is to build a platform for running GP experiments on UCLs' Parallel Computer 'Keter', enabling the GP software to run faster and as a result, yield more accurate data. This project also develops a program for collecting the output data from the parallel machines and viewing it in terms of interactive graphical representation. The output data will be used mainly by researchers to evaluate their GP systems.

Table of Content

List of Figures

List of Tables

1 Introduction

1.1 Why this project

This project intends to guide researchers in modifying their Genetic Programming system to parallelise in UCL Super computer 'Keter' machine and create an application, which would display the output of the parallel computers and produce the necessary results.

This project enables the research students within UCL working on GP systems to get familiar with the college's super computers and the modification involving the GP.

1.1.1 Running on parallel machines and getting results

Because of the nature of GP systems, they tend to be very time-consuming applications. They take much processor power to compute and create generation after generation. Some of the GP system may takes hours if not days to complete.

This project will try to overcome this constraint by distributing the GP system application in parallelised computer. It will utilise some 224 CPU that are some of the best computers in the world to compute the specified GP application and produce the results.

It will also create an application to gather the results of the parallelised computer and display them visually. Because of the multiple results that the parallel computer will produce, it will give a better understanding, accuracy and performance of the GP system. Henceforth it creates a tool for analysis of the GP.

It should be mentioned that it is not the aim of this project to parallelise the internal code of the GP system but to run multiple GP system concurrently on parallel computers.

1.2 Introducing the Problem

One of the necessities of a GP system is getting supplementary results and making it more accurate to evaluate. The current situation at UCL is so that if the researchers want to achieve such results, they would have to run the GP system over and over again and gather the results separately and combine them manually. This will indeed be very time-consuming process, as it will require a great usage of CPU memory and processing power as well as the researcher's time and energy.

To resolve the problem, the GP is given to parallel computers to be run concurrently and will output the results, which will be, collected by graphical representation software. The problem is divided into three sections, each with its own requirements, functionalities and solutions.

First is the modification of the GP system, in which changes to the GP are necessary in order for it to function on parallel computers. There are some issues, which will arise after modification of the GP (i.e. I/O issues). Second is running the modified GP under parallel computers. This will be a program connecting the GP system to parallel computers. Third and last section would be a program for collecting the output data from the parallel computers and displaying them in an orderly fashion.

1.3 Overall Aims

In order to solve this problem and achieve results, two aims have been set. They are:

1. To support the concurrent execution of multiple GP tasks in UCL Super computer 'Keter'
2. To create an application with user-friendly interface to present the output data from parallel computation in terms of interactive graphical representation.

1.4 Objectives

To reach the project aims, they are divided into 4 objectives:

1. To design and implement modification of the GP system in order to function in parallel computers
2. To design and implement a program to control multiple GP tasks running in parallel, with output data collected
3. To diversify the parallel computer output data into several folders within 'Keter'
4. To design an implement a program that accepts the output data of parallel computer and provides automatic interactive graphical representation of it.

1.5 Deliverables

The following are deliverables available at the end of this project.

1. Choose a system and create an account of UCL parallel computers
2. Codes for GP modification
3. Codes for program in 'Keter'
4. Codes for the front-end program
5. Validation of the system
6. The Final Report

1.6 Report Overview:

This report is sectioned into seven chapters. Chapter One, 'Introduction', deals with an overview of the project, it's aims, objectives, deliverables and problem it solves. Chapter Two, 'Background', will deal with the background required on the different topics concerned in this project. It introduces the reader to Genetic Programming and parallel computing.

Chapter Three, 'Analysis and Design', will work through the problem and analyse different solutions available. It will then present the necessary design used in this project. It does so for each of the three sections, the GP system, parallelization and the application. Chapter Four, 'Implementation', will describe how the GP system was modified and made to parallelise. It will also look at the application for displaying the created results.

Chapter Five, 'Testing', will cover the importance of testing in this project and how it was carried out. Chapter Six, 'Validation', validates the output result of the system with previous experiment carried out by researches and presents the outcome.

Chapter Seven, 'Summery and Conclusion', will give and overview of the project, objectives met and challenges encountered. It also looks at future work, and future additions to the project.

2 Background

2.1 Genetic Programming

Genetic programming, developed in 1954 by Nils Aal Barricelli and perfected by John R.Koza, is a concept derived from a genetic evolution in the biological form.

As is seen in nature, humankind as well as other species reproduce by a means of sexual reproduction, resulting in offspring that are genetically related to their parents. The offspring's genes are a combination of genes originated from the parents. This is the concept of evolutionary Algorithm.

Genetic Programming is widely used in financial forecasting and predicting the markets. This is the place where there are a growing number of companies investing heavily in. To put it in few words, GP is used for prediction!

2.1.1 How does it work?

Genetic Programming is a widely used form of evolutionary algorithm. Evolutionary Algorithm uses two Genetic operators in its process.

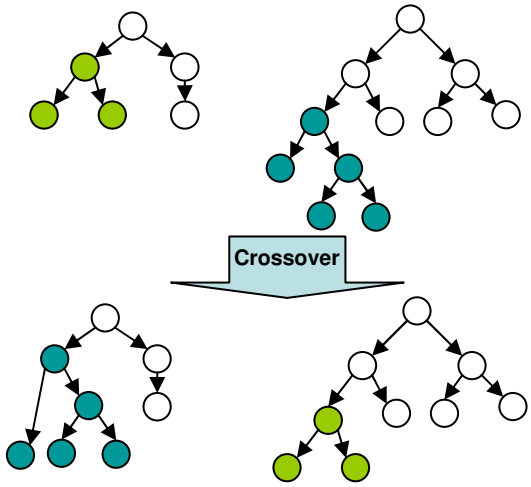

Figure 1: GP System - Crossover

The first one called crossover uses a recombination operation to switch between two randomly selected sub-trees/individuals from one place to another. It has been showed that this will result in more effectiveness of the population and hence the generations that will come later on [3]. As we have seen in Figure 1, this particular operator will effect individual as a whole.

The second operator called Mutation; unlike crossover will look at each individual. It will randomly select an individual (or in some cases sub-tree) and will replaces it with

another individual/sub-tree. The individual that is replaced is a complete new individual [3].

These two operators will ensure that the generations will maintain their supremacy as they are evolved.

One other element that exists in a GP is its Fitness Test. Fitness test exist between generation and its main goal is to perfect the generation as they go along. Each time there is a reproductions and 'parents' produce offspring, there is a fitness test which evaluates whether that child is qualified to go into next generation and reproduce or not. This fitness test will get rid of any individual that are under qualified (weak). Fitness test will enable good and healthy individuals to move onto the next generation and produce their own offspring. This process is very similar to nature's survival of the fittest. It should be noted that the fitness value is based on individual's performance on the GP activity.

Having said this, the GP will start by creating an initial generation. This generation (generation 0) is randomly picked and created. Individuals are made and stored in generation 0. After that, the generation will go through the algorithm to reproduce the offspring and the next generation. This process is shown in Figure 2.

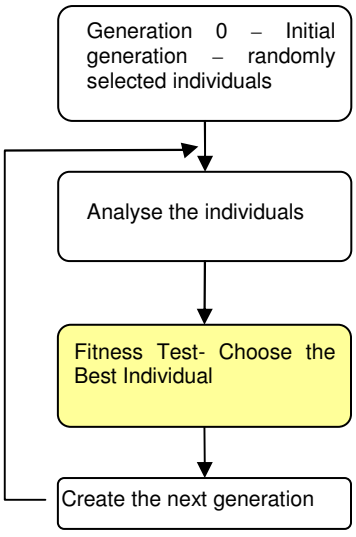

Figure 2: Regular GP System

2.1.2 Current GP

The GP we are using is designed for optimising a financial portfolio given the market's data. It will use certain stocks mentioned in the portfolio to utilising its strategy in trading. [10, 11, 12, 13, 14].

To relate to the definition, each individual in the GP acts as a trading strategy. The computer defines the initial generation randomly. These trading strategies will be tested for their performance on a historical data. If their performance is below the level required they, will be eliminated and disqualified for the next generation.

These individuals will use 22 factors to build up a strategy to trade in the system. These factors are the stock characteristics, which are as follows:

"Closing price, Price Momentum, Volume, Price (Cash), Book price, P/E Ratio,30-day MA, MA Changes, Volatility, Dividend Yield, Earnings on equity, BVPS, Market Capitalisation, Changes ROE, Revenue Growth, cash/Share Yield, Adjusted Dividend Yield, adj EPS, 1Y Earn Growth, Equity Asset, Z Factor, Cps Dps"

Below is a sample of an individual who had the top performance after 25 generations.

```
(+ (Log (- (+ (Log Price_Mom) (Log (Log (* (/ (- (- (- Market_Cap (/ Market_Cap MA_30_Day))
Market_Cap) (/ (+ Equity_Asset Book_Price) (/ Equity_Asset (+ Equity_Asset Book_Price)))) (-
CPS_DPS (Pow Close Divi_Yield))) Z_Factor)))) (- (+ BVPS Change_ROE) (- CPS_DPS Earn_Growth)))) (+
(* (* (+ (- CPS_DPS Earn_Growth) Divi_Yield) (Log Equity_Asset)) (Log (+ (+ (* (* (+ BVPS
Share_Yield) (Pow (+ Adj_Divi_Yield BVPS) Volume)) (- Change_ROE PE_Ratio)) (- (Log Divi_Yield)
(Pow (- Z_Factor (/ Z_Factor MA_Changes)) Divi_Yield))) (* (/ (+ (Log (* (/ (- (- (- Market_Cap (/
Market_Cap MA_30_Day)) (- EPS Divi_Yield)) (Pow Close Divi_Yield)) (- CPS_DPS (Pow Close
Divi_Yield))) Z_Factor)) (- (- (Log Divi_Yield) (+ Equity_Asset Book_Price)) Equity_Asset)) (+ (Log
(- (+ (+ (+ Book_Price Book_Price) (/ Market_Cap Share_Yield)) (* Z_Factor Earn_Growth)) (- (+ BVPS
Change_ROE) (/ MA_30_Day Market_Cap)))) (+ (* (* (* (+ BVPS Share_Yield) (Pow (/ Z_Factor
MA_Changes) Volume)) (Pow (/ Z_Factor MA_Changes) Volume)) (Log (/ Change_ROE Market_Cap))) (* (Log
(/ (* Volume Market_Cap) (+ Adj_Divi_Yield BVPS))) (Pow (+ ERC[f1147453440|915.0|] Price_Cash) (Pow
CPS_DPS Earn_Growth)))))) (Pow (+ ERC[f1147453440|915.0|] Price_Cash) (Pow CPS_DPS
Earn_Growth)))))) (Log Rev_Growth)))
```

2.1.3 GP Phases

Genetic Programming that is being used has two phases.

The first phase is called the **training** phase. This is the phase in which generations are being produced over and over again by using a sample historical data provided by the user.

These individuals would be testing their strategy on a historical data. Based on their performance , they will be given a single integer value (in some cases a vector of values)which is called the 'Fitness Value'. These fitness values would be criteria for the fitness test to see whether the individual is qualified to proceed to the next generation or not. It should be noted that the next generation, and generally all generation in the training mode uses the same historical data over and over again. As it was said before, they will constantly perfect each other using a Fitness function, eliminating bad and under performing individuals

In another words, the individual conducts its own strategy (experiment) over a number of months and eventually over the years. Each individual's performance will be measured by going through a fitness test. It will determine the continuality of the individual to the next generation. This way the generations will constantly perfect each other.

After the training has been done, the application would select the best individuals that have survived. This will lead to the second phase called **Validation** phase. This phase will use these best individuals see how their strategies will perform under a new set of data which has not been shown to them before. This is considered the most important phases in evaluating the GP system.

For example, the training phase uses the data over a 12 month period to perfect the individuals. Then the validation phase uses the next 6 months of data to test the individual.

2.1.4 Evolutionary computing techniques

Additionally evolutionary computing has two techniques called Single Objective and Multi Objective. Both of these techniques are similar in the fundamental aspect of evolutionary programming.

2.2 Parallel Systems

Two different types of computation are widely used in these systems. The first would be the so-called serial computation. This traditional method of computing is for single CPU systems. Single CPU systems are able to carry out single operation at a time. In serial computation, the problem is broken into series of instruction and executed one after another, illustrated in the Figure 3.

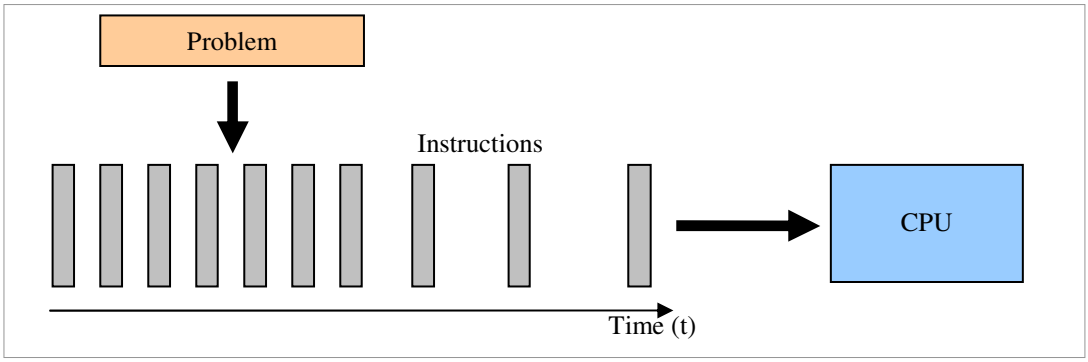

Figure 3: Single CPU system

The second type of computation would be parallel computation. Parallel computation is similar to serial computation but the problem is broken down into series of tasks. These tasks do not rely on each other's output and can be computed separately. They are distributed and given to an array of CPUs. In effect, each task is performed by several single CPU systems and thus the overall problem is solved much faster. The speed of the process depends on the number processors available and how well the system can break down the problem into parallel running tasks. This is illustrated in Figure 4.

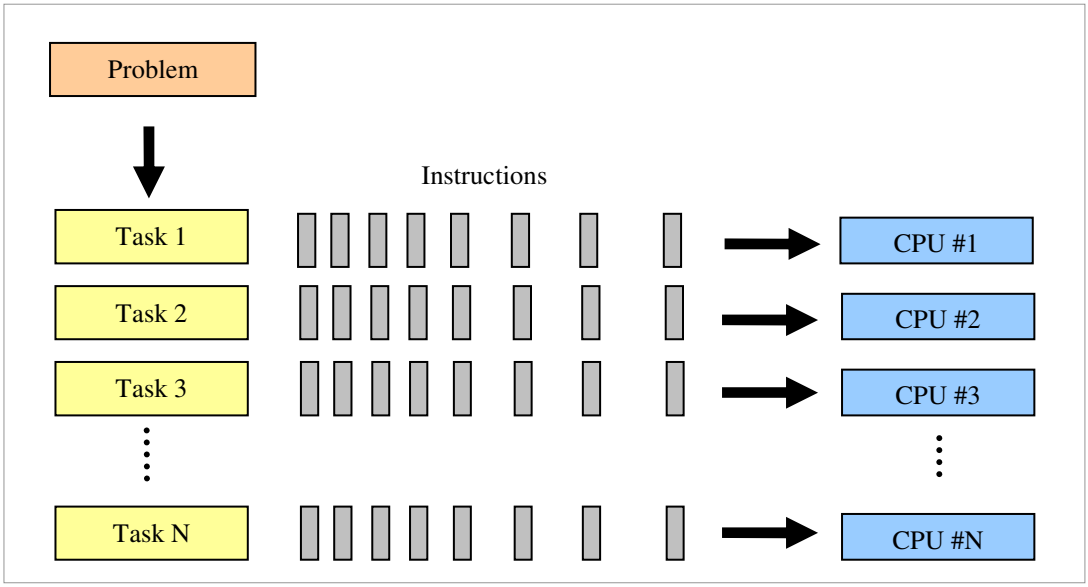

Figure 4: Multiple CPU system

It should be noted that after the Parallel System gets the job from the user, it will transfer it to the 'Job Distributor' which will then allocate the job to the computers available.

Parallel System and Computing has been used in variety of complex system and have aided scientists to reach some of the most important conclusions. Amongst them are cancer research, aerospace system modelling, weather and climate change and many more. In fact, the BBC along with Oxford University ran a climate change experiment, in which computers around the world would download the program and take part in running a model.

2.2.1 What Parallel architectures do UCL offer?

UCL currently offers six different parallel systems. These architectures are mainly for calculating computational problems and handling complicated equations. UCL researchers are free to use these services [6]. These services are as follows:

2.2.1.1 Access Grid

Facilities for virtual collaboration

The Research Computing Access Grid studio is a set of computers that mainly supports remote visualisation, distributed meeting and distance education. With its large display, screens it enables the user to experience face-to face meetings. [6]

2.2.1.2 C^3

For advanced batch style

Central Computing Cluster (C^3) is a Linux based cluster which is optimised for low I/O jobs. These Processors are Ideal for serial jobs and parallel computing. C^3 currently has a variety of software, compilers and specialised library installed on their systems. These Compilers are namely JSDK, GCC, Perl, and Python. C^3 has 96 IBM Dual Intel Xeon Processors running at 2.80 GHz. [6]

2.2.1.3 Condor

High throughput commodity computing pool

Developed by University of Wisconsin-Madison, Condor is a series of cluster with around 1400 PC s equipped with windows platform ideally for high quantity of similar jobs. Each individual is capable of processing several jobs at a time and thus with a large cluster, Condor can manager hundreds of jobs round the clock simultaneously. On the statistics, 20,000 jobs are being submitted onto Condor every month. It is said that UCL Information System Cluster (WTS) are currently using many of the machines. [6]

2.2.1.4 Prism

High performance visualisation resources

Prism, Capable of processing high details graphical processes is a high performance visualisation tool. With an exceptionally high graphic card installed in each of the processors, Prism can allow the viewers to wear 'passive polarising glasses' to see an impressive 3D Image. [6]

This resource is mainly used by chemistry department to calculate high detailed image processing jobs. [6]

2.2.1.5 Altrix

High performance computing

Altrix, is one of the facilities for actual parallel computing, the other being Sun Cluster 'Keter'. This architecture is ideally for high performance computing processes. Unfortunately Altrix only supports C and C++ (along with FORTRAN and MKL) Compilers and does not support Java. These 56 Processors have limited libraries and are only suitable for a limited number of jobs which requires a great deal of CPU memory and scalability.[6]

2.2.1.6Sun Cluster ‘Keter’

Serial and parallel computing

Sun Cluster ‘Keter’, the other facility for parallel computing. It is a comprehensive system with wide range of libraries, tools and resources for parallel and serial computing. Keter is a 224 Sun Cluster servers with Solaris as the operating system. It has the ability to run jobs that require java and is ideal for I/O profile application. [6]

3 Analysis & Design

In this section, the materials refers to problems about the GP system, and how it should be customized and tailored to fit the requirements of the Sun Cluster 'Keter' System. It will also deal with parallelisation of the GP system and the front-end application requirements in order to provide an interface for the user.

3.1 Problem with current GP system

Generally, there is a problem with all the applications that output any type of file including current GP system; they are simply not suited for parallelization. Given the fact that they are not specifically been written in a parallel manner.

One of the problems encountered with the GP system is its Input Output (I/O). This can be explained by a simple example below.
If the GP System runs on one machine, it will eventually generate the necessary output. The output data will go to a directory specified by the application.
However, if the program concurrently ran on parallel machines then it will have difficulty outputting files. This is because the output data will go to the directory specified by the application each time it ran and ultimately it will be overwritten.

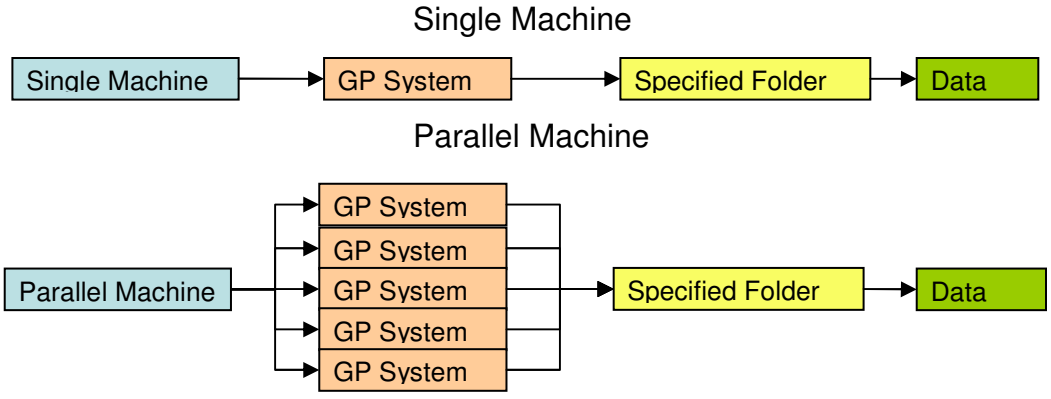

Figure 5: GP Under Single Machine vs GP under Parallel Machines

As we can see in Figure 3, in both cases if the GP system is not modified, we will only output one data. In the parallel machine case, the data would be the output of the last GP since it constantly being overwritten.

One of the modifications that can be useful in this GP experiment was outputting the maximum data that the class deals with. In this GP system, a few statistical data of a single generation is shown. For Example the Final Statistical File, an output of the current GP, currently shows the Best, Average and Worst of the three Standard Deviation, Sharp Ratio and Annualised Return value for each generation. However, it only showed the average of three values and best of two values. To fully utilise this, there should be an addition of Best and Worst was added to all three values. This will not only optimise the output, but it will also allow the front-end application to display more result and present more information to the user, resulting in a more accurate analysis by the user.

Furthermore, when creating a GP system, it is important to design it such that it would run on every Operating System. If the GP system was previously designed and implemented in windows, then it would be essential to change the design if we want to run it on UNIX or a parallel System. This would become clearer when the GP system refers to a directory or a file within a specific Operating System.

In addition, when running the GP system on parallel computer, it is vital to provide it with class path and necessary environment in order for it to run efficiently.

3.2 Parallel System

Once the GP system is modified for the Sun Cluster 'Keter' System, we will try to run the program on the machines.

Sun Cluster 'Keter' is one of the two facilities provided by UCL to support Parallel Computing – the other being Altrix. The main reason why 'Keter' was chosen is the fact that it supports java and its mechanism. 'Keter' is also widely used because of its environment and its speed. With 224 Sun Cluster processors, it can rapidly compute jobs.

In order to be familiarised with 'Keter' System and its environment we must get access to the system. An application should be made to the Computing Coordinating Manager; He/She will then forward it to the System Administrator to process it.

Once the application is complete, we can then have access to the system. Keter environment is much like UNIX environment. It functions with all the commands that UNIX usually have. It will also have the special command for the 'Keter' System settings (i.e. Submitting Jobs).

In order to the submit jobs to Parallel Architecture 'Keter' we must first create a runscript. A runscript is a script that is used by Sun Cluster 'Keter' System to distribute it to other computers using a Job Distributor.

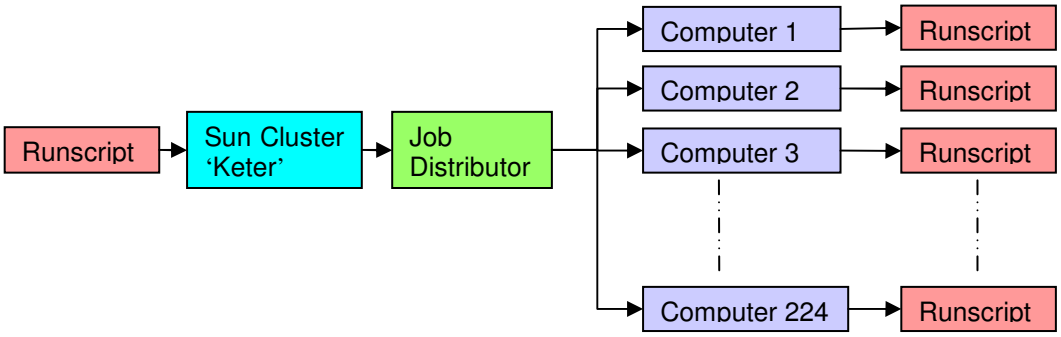

Figure 6: Parallel Computer 'Keter' Overview Runscript Given

The Job Distributor's responsibility is to allocate jobs to each computer depending on their availability. An overview of the process is seen in Figure 6.

When designing a runscript, we must define the main Java class as well as its class path. We would be specifying where to run the job and how to run it. We can also specify how many jobs we would expect 'Keter' to run for us, though this is an option as we can also use this command when submitting the job.

3.3 Application

Having analysed the back end of the project, we now move forward to the front-end to see what sort of application should be built in order to receive and analyse the necessary data from the GP system and display it to the user.

3.3.1 What kind of application would be useful?

In order to best fit the criteria for displaying the GP data, a user-friendly, fast and reliable application that can display graph data accurately is essential.

3.3.1.1Specification of the application

To explain in detail, the 'back-end' application is divided into two main sections:

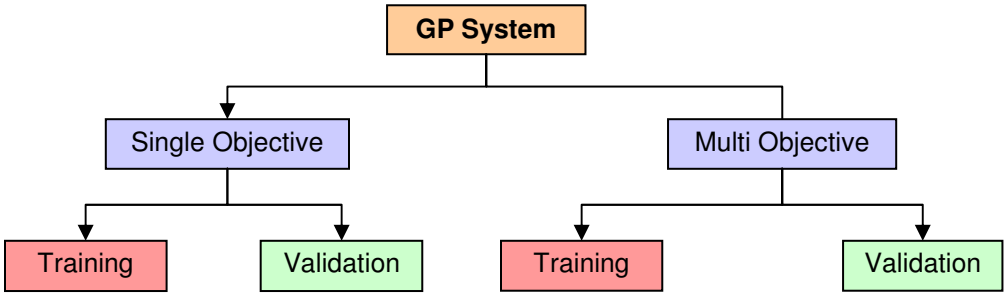

Figure 7: Current GP System Overview

The first section is Single Objective Training case where the generation in the GP system consistently perfect each other by testing on historical data and going through a Fitness test. However, in Validation Mode, a selected number of Individuals are put to test to see how they would perform given new historical data.

In this section, the front-end application is concerned with two types of files as its input; each type of file can be either Training or Validation.

Looking at Figure 8, the first type of file is the so called *'Population File'*, which would contain the individuals of each generation. Each individual will hold three values of Annualised Return, Annualised Standard Deviation and Sharpe Ratio. In Training mode, the *'Population File'* contains a list of Generation with a Generation Separator string dividing each generation.

As for the Validation Mode, there will be a selected number of individuals chosen by the application from the Training Data. Therefore, it would not have any generation or Generation Separator string, but only the chosen individuals.

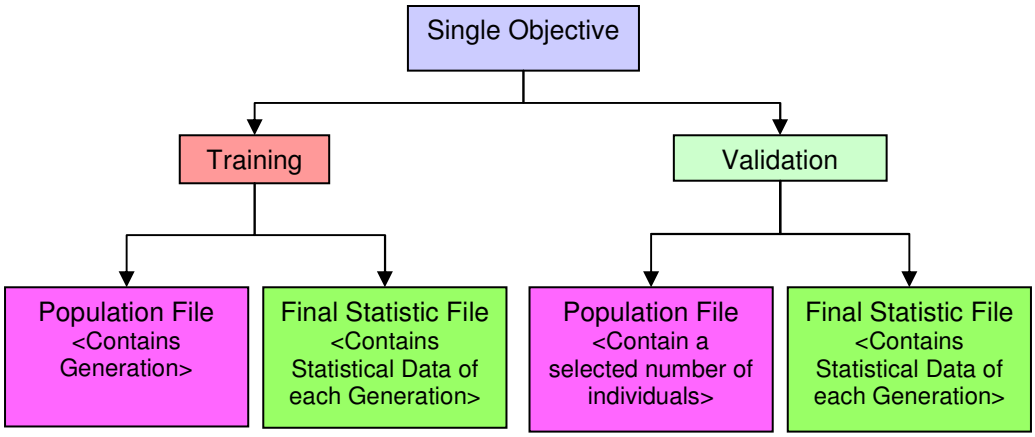

Figure 8: Single Objective Overview

The second type of file that would be imported and recognised by the application is the *'Final Statistical File'*, which contains the statistical evaluation calculated for each generation. It will hold the Best, Average and Worst values for each of the generation's Annualised Return, Annualised Standard Deviation and Sharpe Ratio. The *'Final Statistical File'* would be similar for both Training Data and Validation Modes.

Overall, the application should be flexible in allowing the user to import its own file as well as locating the parallelised files.

When analysing the parallelised files, the user locates folder which contains the parallelised file produced GP System in by Sun Cluster 'Keter' and uses a searching algorithm to locate the files to evaluate it, structure it and display it. A file structure is shown in Figure 9.

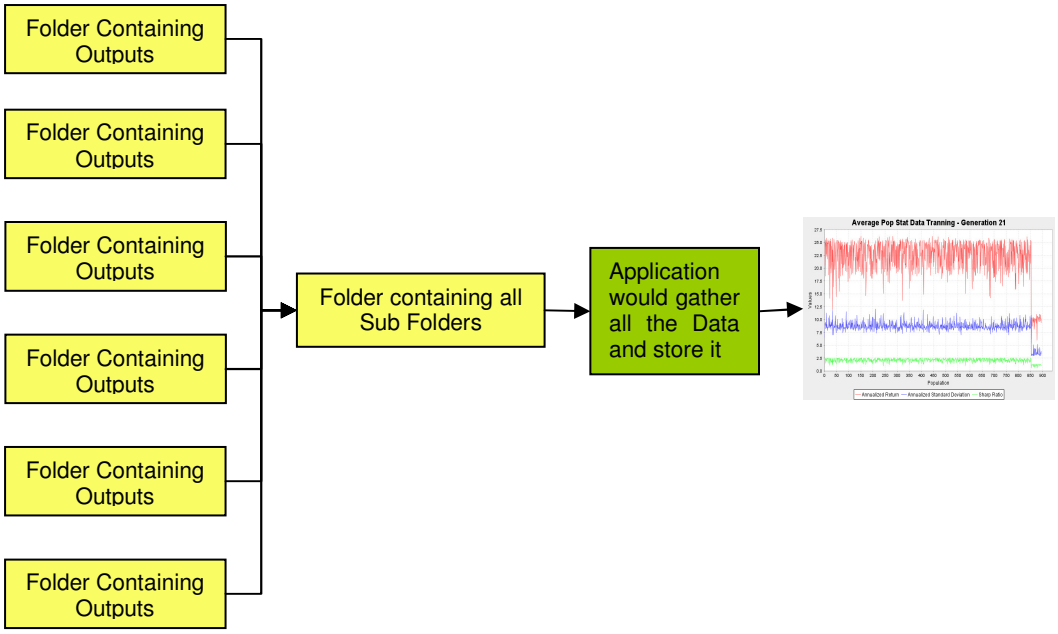

Figure 9: Parallel Folders to Application

Once the parallelised files are retrieved, the application will use a mechanism to structure each type of file, and store it into a separate database. After doing so, it will combine all the structural databases together and calculate a mean from them.

The second section of the GP System is Multi Objective mode.

In this section, the front-end application would be taking four types of files to analyse since the *'Population File'* in Validation and Training modes would defer from the previous ones.

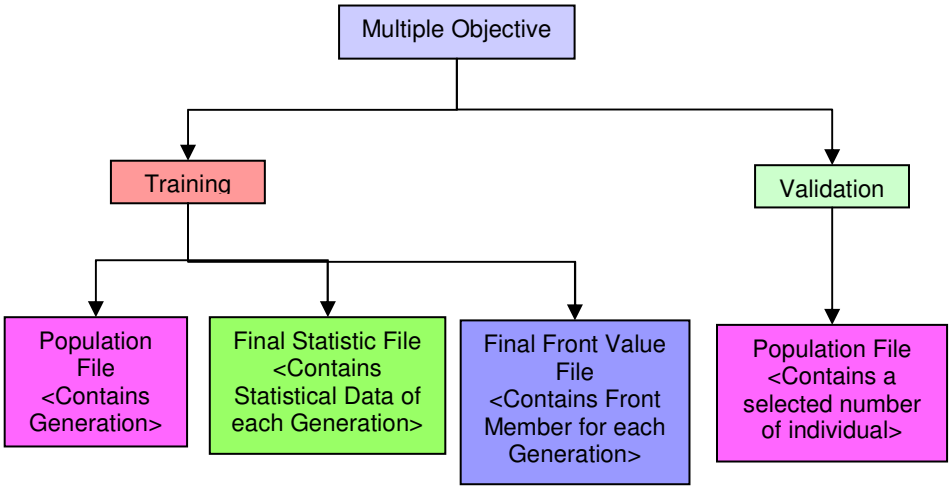

Figure 10: Multiple Objective Overview

Looking at Figure 10, the first type of file is *'Population File'* for Training Mode. The population is similar to the *'Population file'* we have seen in the Single Objective – Training mode. The file contains all the individuals of all the generations. Each individual holds value of Annualised Return, Annualised Standard Deviation and Sharpe Ratio. A Generation Separator String also divides the Generations.

The second type of file we see in the above graph is the Final Statistical File for Multi Objective Training mode. This file is again similar to what we have seen before in Single Objective – Training mode. It contains the statistical evaluation calculated for each generation. It will hold the Best, Average and Worst value for each of the generations Annualised Return, Annualised Standard Deviation and Sharpe Ratio. Therefore, it will hold nine values for each generation.

In addition, the last file is the *'Population File'* for Multi Objective Validation Mode. The file contains a selected number of individuals from the Multi Objective Training mode. These individuals will each contain three values of Annualised Return, Annualised Standard Deviation and their Sharpe Ratio.

Once the application is aware of these types of files, it will then give a choice to the user to select their own single file or to extract the output of Sun Cluster 'Keter' Machines and feed it into the application.

It is important to mention that the application is design so that it will take the format of which the 'Keter' System output and evaluate it as it is without the user modifying it. This would save a great deal of time, as the user would not have to modify any data.

A Time Series Graph best presents all of the Data that the GP System produces, regardless of their type and whether they are producing in Single Objective or Multi Objective modes. This is mostly because the specific value are not very much indifferent to us but the values as whole in long term over generations is vital, and they would be best presented with a line graph.

3.3.2 Sections

The 'front-end' application where the user communicates is divided into four main sections.

3.3.2.1 Section I: Single Objective – Single Run

This section adds flexibility to the application since it is initially designed for analysing parallelised output of the Sun Cluster 'Keter' System.

It will allow the user to browse a single file and specify whether it is a *'Population File'* or *'Final Statistical File'* with the choice of Training or Validation. After selection, the application would read the files and store them in a structural database. Having done so, it will use a Plug-in to display the results in terms of line chart and graph. The user can also specify which generation it desires (for 'Population file') and which of the nine values in 'Final Statistical File' they want to view.

3.3.2.2 Section II: Single Objective – Multiple Run

In this section, the user would specify the directory along with the file names and a string separator, in which the parallelised files are located. Then the application would use a searching mechanism to locate the files, classify them as Training or Validation, structure them into a database and demonstrate them. The user can also specify which generation it desires (for 'Population file') and which of the nine values in 'Final Statistical File' wants to view.

3.3.2.3 Section III: Multi Objective – Single Run

In this section, the user can specify the location of either the 'Population File' or the 'Final Statistic File' or the 'Final Value file'. The application would use a reader to read the files, store them into a structural database, and demonstrate them in terms of graph.

There is also a choice of Training or Validation mode that can be specified by the user. Since the Validation mode only has 'Population File', the 'Population Control Panel' is the only enabled control panel there.

3.3.2.4 Section IV: Multi Objective – Multiple Run

The section similar to Single Objective – Multiple Run section deals with the parallelised files created by Sun Cluster 'Keter'. The user will locate the folder that the parallelised files contain, specify the file names and Generation Separator String, and the application would use a searching algorithm to locate these files. Once located, it will use a reader to read the files and structure them into a database explicitly defined for Jigloo Plug-in, which is responsible for displaying the graph.

3.4 Summary of Requirements

- Modification of GP system to function on parallel computers
- A program to enable the GP system to run on parallel computers
- The user should be able to run as many jobs it requires.
- Diversify the output in different folders when jobs are ran on 'Keter'
- A program to receive the output data and present them in terms of graph
- A user-friendly interface for the front-end application
- An OS independent program
- The user should be able to choose Single Run/Multi Run
- The user should be able to display single generation specified (in the Population File)
- The user should be able to select a specific trend within the Final Statistics File.
- The user should be able to select between training and validation
- The user should be able to select between Single Objective and Multi Objective.
- To Modify Current GP System and optimise it for parallelisation
- Creating the means for the GP system to run on 'Keter' Machine
- To create a software platform which the results of the GP system can be displayed and analysed

3.5 Use Cases

Use Case ID:	1
Use Case Name:	Submitting Jobs
Created By:	Ali Adeli

Actors:	User
Description:	To Submit Jobs into 'Keter' Parallel Computers
Trigger:	
Preconditions:	To log on to 'Keter' System.
Post conditions:	Job is submitted to 'Keter machines, and It will be processed.
Normal Flow:	1. The user logs on to the system 2. The user will access the folder containing the Runscript file 3. The user will then type "*qsub runscript.sh"*

Use Case ID:	2
Use Case Name:	Retrieving Population File Data and Graphs
Created By:	Ali Adeli

Actors:	User
Description:	To Retrieve Population File and Display its Graph
Trigger:	The front-end program should be running
Preconditions:	All the functionalities relating to the program must be working.
Post conditions:	A Graph will display Generation 0 of the Population file.
Normal Flow:	1. The user will click the browse button 2. the user will locate the Population File 3. the program will then write the pathname of the file into a text box 4. the user will press the retrieve button 5. the program will display Generation 0 instantly

Use Case ID:	3
Use Case Name:	Retrieving Final Statistics File Data and Graphs
Created By:	Ali Adeli

Actors:	User
Description:	To Retrieve Final Statistics File and Display its Graph
Trigger:	The front-end program should be running
Preconditions:	All the functionalities relating to the program must be working.
Post conditions:	A Graph will display The Final Statistics File. This Graph will be blank at first, but when the user selects a specific trend, it will show it.
Normal Flow:	1. The user will click the browse button 2. the user will locate the Final Statistics File 3. the program will then write the pathname of the file into a text box 4. the user will press the retrieve button 5. the program will display a Blank Graph • The use will then select a specific trend to display (i.e. will select Average and Sharpe Ratio) • It will display the graph relating to it.

Use Case ID:	4
Use Case Name:	Retrieving Multiple Run Parallel Files (Population File) and displaying the Graph
Created By:	Ali Adeli

Actors:	User
Description:	To Retrieve Multiple Population File and Display its Graph
Trigger:	The front-end program should be running
Preconditions:	All the functionalities relating to the program must be working.
Post conditions:	A Graph will display Generation 0 of the Population files.
Normal Flow:	1. The user will click the browse button 2. The user will locate the directory which hold sub-directories containing Population Files. 3. The program will then write the pathname of the directory into a text box 4. The user will press the retrieve button 5. The program will display Generation 0 instantly

4 Implementation

Implementation of this project is also divides into three sections. First is the modification of the written GP code, which required understanding and knowledge of its classes. Second creating the means for the GP system to be concurrently executed on UCL's 'Keter'. Finally, to create a software platform in which the results can be displayed and analysed.
Each section has its own interfaces and programming languages to work with making it necessary for separate examination.

4.1 Modifying GP Structure

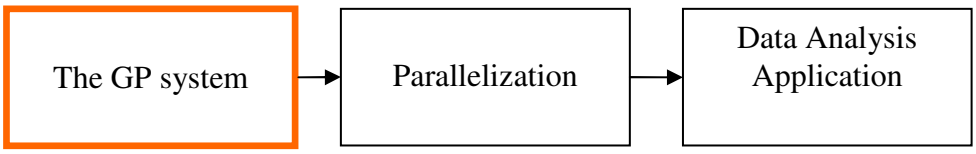

Figure 11: Project implementation subsections – GP System

In order for the GP System to work with 'Keter' Parallel Computing Architecture, There had to be some modification made to it.

4.1.1 Modification of classes

As we described earlier, in order to modify the GP system to work with the Cluster Machine, we examine the structure of how the application operates and what kind of output mechanism is used.

In general, when modifying a GP system, we would be looking at two types of classes. The first is the class(s) that produces 'seed number', and second, the class(s) that produces the output files.

In this GP system, a class called Evolve.java created the seed number. The seed number is generated through 'Time stamp' and uses the time of the system to calculate

the seed number. We would later use this number to create folders with the numbers as the folder names. This would always give us a new folder with an unused name. It should be mentioned that this number is always unique (not randomly generated) and can never be the same even if it is run in parallel[1]. This would be a very innovative way to over come the I/O Issues, as it would redirect each file to its own directory.

After the classes were located, they were modified so that the seed number is made public and would be accessible and visible to other classes. The seed number in the class(s) is later utilise to produce the output files.

Once the seed number is made public, the classes that produced the output file of the GP system would be located. In this GP there are two classes which would particularly be responsible for outputting the GP data.

The initial and most important class that is responsible for most of the data is called InvestSim.java (short for Invest Simulator). This class is very significant because it is also accountable for changing modes of the GP system from Single Objective to Multi Objective and toggle between Training mode and Validation mode. The changes made to this class is critical as the sensitive GP data would be generated through it.
In the InvestSim class, before making any changes, a main directory called JobData was created and all other directories that are later created were dumped inside this. This will prevent the user from confusing the parallelised file with the other GP files. Figure 12 illustrates this architecture.

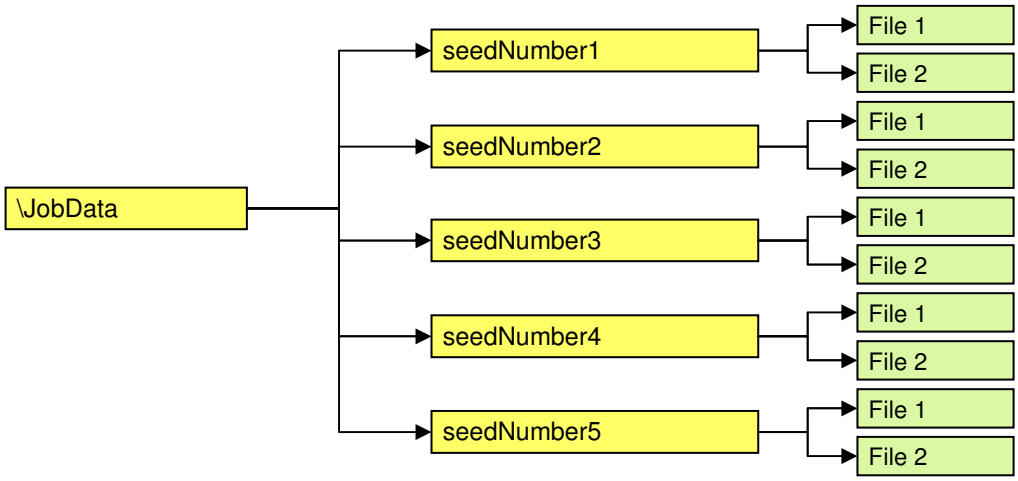

Figure 12: Parallel Files -File Distributed

Since these changes must happen instantly upon calling this class, the codes to create them were added to the constructor. In addition to creating the folder called JobData in

[1] Through out the experiment, there was not a single case where the numbers and the folders overlapped. This proved that the number would be always unique, because two jobs would never precisely start at the same time.

the constructor, the seedNumber generated by Evolve.java class is used to create a directory with the directory name as that seed number. This directory will be used to store all the GP outputs for that run.

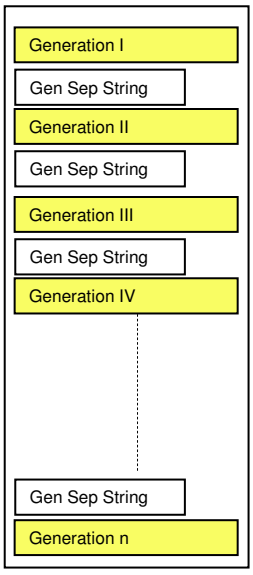

Figure 13: Population File Outline

Once the folders are created, we will then identify were the data files are actually being outputted. Once located, the seedNumber imported from Evolve.java, the seedNumber generator, is used to re-direct the files to the necessary folder. It should be mentioned that this diversion should be made to all the data files that are being outputted by the application.

In addition, there has to be a modification done to the class which generates and outputs the file containing individuals of the population (Population File). In this case, it would be the same class that generates all the output files, InvestSim.java.

Once the method that generates this *'Population File'* is located, we can then add a Generation Separator String that would simply differentiate between each Generation. It will ensure that the front-end application would have no trouble reading the generations. In Figure 13: Population File Outline , the Generation Separator String is between each generation.

Another way of doing the separation is adding one column to the *'Population File'* writing the Generation Number for each individual. This would not be a good idea, as it will not only increase the time it takes to load the files into the front-end application, but it will be ineffective, as it is already known to us that the generation are increment.

4.1.2 Modification of the main GP code:

The modifications mentioned above will result in a complete parallelisation of the GP system, but there can be various additional configurations to the GP system that will allow the front-end application to get additional data.

One other modification mentioned earlier is outputting the maximum data that the class deals with. In order for this to happen, we must look what data is generated by the classes and how can it be optimised. One of the output data that can be optimised is the *'Final Statistical File'*. This file shows the average and sometimes the best of the three, Standard Deviation, Sharp Ratio and Annualised Return value for each generation. This was in fact optimized by displaying all three possible values for each generation.

It should also be mentioned that one of the modification that must be done to the GP system is the file path modification. Since most of the current GP system solely run under a specific Operating System, it should be ensured that when using a file path, instead of '\' or '//' for other Operating system, a "File.Separator" should be used to guarantee that the file path is correct under any operating system. This small detail caused many problems during the implementation.

4.2 Parallel System

Figure 14: Project implementation subsections – Parallelization

Having modified the GP system, we will then try to execute it on the 'Keter' Machine.

4.2.1 How does 'Keter' work

In order to access the 'Keter' Machine, we acquire an account with the system. When doing so, we must send an application to Research Computing Coordinating Manager, which will then be forwarded it to the System Administrator to process it.

Once an account has been created, we can use 'SSH Secure Shell Program' to access 'Keter'. It is fair to mention that 'Keter' environment is much similar to UNIX environment with addition of extra commands for parallelisation.

4.2.1.1Creating runscripts

In order to run the same application parallel to each other, the user must create a so-called 'Runscript'. This is a script, which with its unique style allows 'Keter' system to locate, read and execute the program as many times as the user requires.

In another words, runscript is a .SH file, which is read by the 'Keter' system, it will then be forward it to the jobs distributor. The job distributor will then allocate this job to the Cluster machines with this runscript.

After directing to relevant path, it allows the system to run the necessary command several times in parallel.

It should be noted that when creating runscripts, it is vital to modify it under Keter environment. This is mostly because if it is modified under for example Windows, some unwanted characters will be added without the user knowledge (i.e. the character (^M) at end of each line).

4.2.1.2Submitting jobs

After creating the necessary runscript, the user will then have to submit it to the 'Keter' system. This can happen by going to the relevant directory that contains the runscript file. Once in the directory the user will then type

```
qsub ./runscript.sh
Your Job 232 ("name") has been submitted
```

This command will automatically place the job in the queue. After it has been placed in a queue, the system will then give a job ID to process it.

Furthermore, if the user wants to submit several identical jobs and actually utilise parallel computing he/she must type the following command.

```
qsub -t  1:10:1 ./runscript.sh
Your Job 232 ("name") has been submitted
```

Once you type this command, Keter will automatically put your job in the queue for processing your query. Keter will also allocate a number to this Job as a whole and each sub-job a number of its own.

Since each sub-job has a number allocated to itself, this command will start job 1 to job 10 in steps of 1. So this command will therefore execute **10 jobs** (1,2,3,4,5,6,7,8,910).

If you type **`qsub -t 1:10:2 ./runscript.sh`** it will therefore execute five jobs (1, 3, 5, 7, 9).

It is important to note that these numbers are for the convenience of the user and will not affect the number of jobs you submitted. Therefore, 5:10:1 will be the same as 15:20:1, and the only thing that is different is the sub-job number.

'Keter' system works in first come first serve basis. Therefore, the duration of the job depends very much on several factors. First when the job is submitted, if the queue is lightly loaded, it will take a short time to process it and produce the necessary result, but on the other hand, if the queue is heavily loaded, it might take up to several days to process the job. Second, if the submitted application takes a very long time to process, then a similar proportionate time should be expected from the 'Keter' system.

If the user wants to monitor the status of the job, the following command can be used:

```
qstat
```

Similarly, if you wish to kill one of the jobs, the following command can be used:

```
qdel [jobID]
YourID has deleted job 232
```

4.3 Application

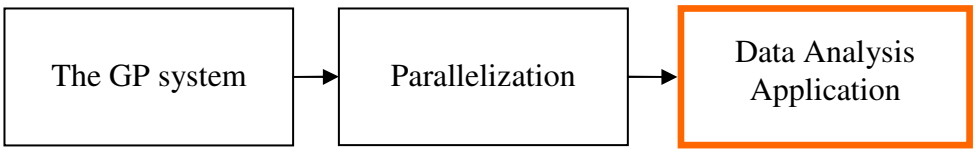

Figure 15: Project implementation subsections – Application

Before getting into the details and implementation of the application, lets consider it's purpose and aim. As seen in Figure 15 this application is the third and last part of the implementation and acts as an interface. The aim of this application is to read and reorganize the data generated by the UCL Super Computers (Sun Cluster 'Keter') and combine, analyse and display the result.

4.3.1 Programming Language

To implement this application several languages where considered and the following was chosen:

4.3.1.1 Java - Ideal Programming Language

Java, almost certainly is the most widely used language in the industry. It currently powers more than 2.5 billion devices and has attracted over 5 million software developers worldwide[2]. This object oriented programming language is platform independent. In another words the written software can run on practically any other platform and can be modified to run within web browsers.

As part of the requirements for this project, it was advised and encouraged to write the program in Java. This was mainly because Java is platform independent and it was widely used in first and second year of computer science courses, therefore the students were very much familiar with it.

4.3.1.2 Platform/Plug-In/Packages used

Eclipse

Eclipse, originally developed by Eclipse Foundation is an open-source software which is platform-independent framework to develop application such as Java. This

[2] http://www.java.com/en/about/

framework is widely used throughout the community and has consistently been providing new updates and new plug-ins.

Although a strong tool, there are some problems to the application that was encountered. One of the problems were the application simply takes up too much memory. In another words, if the application is running for a period of time, it caused the computer to slow down and sometimes crash. In addition, Eclipse has been very slow when running various classes and sometimes will not respond. However, this might be a weakness, but it would certainly not discourage anyone from using it.[7]

There are similar frameworks like Eclipse, which it is widely used. They are namely NetBeans, JDeveloper and JBuilder.

GUI in Java

Creating a Graphical User Interface (GUI) is unfortunately not the strength of Java. Though being a strong dynamic language, it is not capable of demonstrating a very strong user-friendly GUI to the user. This is mostly because of its specification and platform independent feature.

Not surprisingly, there are Plug-ins out there to aid the software developer in creating GUI easily and accordingly. One of the plug-ins provided is jigloo.

Java plug-ins – Jigloo

As one of the GUI plug-ins in Eclipse, Jigloo is an open-source publicly distributed library that supports a very straightforward creation of Graphical User Interface and is labelled as a 'GUI Builder'. It currently creates GUI with Swing or SWT. [8]

One of the features that is considered a plus for Jigloo is its dynamisms towards the software developers. It is highly customisable and very fast.

Jfreechart

Jfreechart is a free Java class library for generating charts. This java library would allow the user to benefit from wide range of chart. These are namely bar charts, line charts, pie chart, time series charts.[9]

One of the charts that were used in the application was time series.

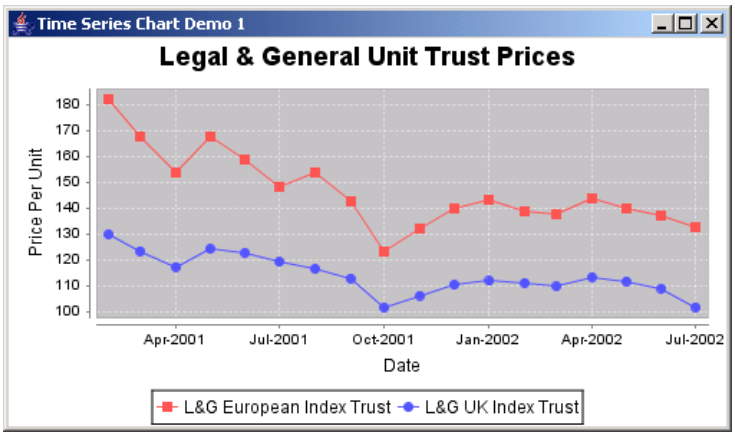

Figure 16: Example of Jfreechart graph

This is because of it is evidently demonstrate to best ability the data that the program wants to display.

4.3.2 Input File Type and Graphical Representation

Before going into the sections of the application, the input file type and chart data structure needs to be reviewed. The outputs of the GP systems are Comma Separated Value Files (CSV). The files are readable both by excel and other spreadsheet programs. Yet they cannot display the results in an orderly manner. As it will be described in the next section, there are several CSV files for different condition, whether it is single objective – validation or multi objective – training.

Each of these conditions needs its own separate extraction and processing, which will be looked at in the next section. However, after processing they need to be structured differently to be ready for display.

It is very important to mention that this process for displaying the data generated by the GP system to the graph ready data by the application. After the data has been organised in a structural way, it will then be transformed into classes that is understandable to the graphical section of the application. This section will use the data to create graphs.
The graph created by the application contains the trends and a legend. It will also zoom in, zoom out and save the picture as well as print it.

4.3.3 Sections of Application

As said in the previous sections, the GP system is capable of performing in two Modes, Single Objective and Multi Objective. One of the aims of this project was to develop an application, which would collect the output generated by the parallelised machines and display the analysed data.

Before going into any detail, it should be noted that the application does have the ability to analyse single runs of GP system where the GP will be running on a single machine as well as multiple runs of GP's where there would be parallel computers, concurrent runs and multiple data's.

Below is a table listing different possibilities and outcomes that the application accounts for each possibility, i.e. Single Objective-single run-validation, produces several files with different data types which will be discussed later on.

Modes \ Runs	Single Run	Multiple Runs
Single Objective	Training	Training
	Validation	Validation
Multi objective	Training	Training
	Validation	Validation

Table 1: Single/Multi Objective Modes

4.3.3.1 Part I: Single Objective

In single objective mode, each individual within the generation has a certain fitness value, which is a single integer. There it can trivially be indicated if one individual is fitter than other is. This comparison will determine the best, average and worst individuals.

In order to convert what is produced in the GP system to what is seen on the graphs, we will go through several sections. The first section is retrieving the data produced by the GP system. This is done either by the application using a search algorithm to locate the files or by the user selecting it. After the data is retrieved, the application uses a reader to read the files and store it into a structured data. This data would be tailored so that it can later be used by the graphical section of the application.

A similar diagram can be seen here. This is when the user selects a specific GP output for the program to analyse.

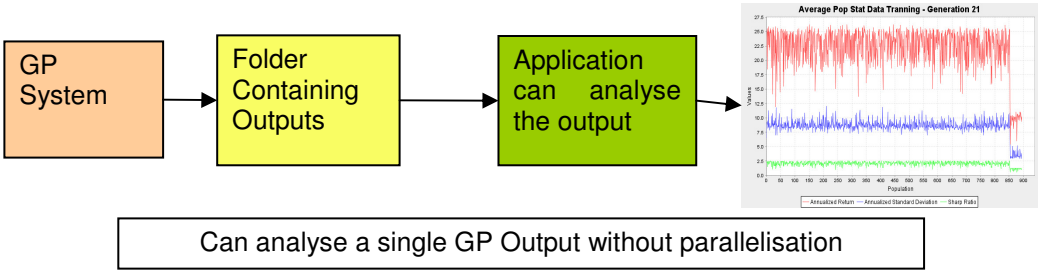

Figure 17: GP Single Run

The GP system that is being worked on produces two forms of files on Single Objective mode.

The first file contains the individuals of each generation, known as the population file, and the second file contains the statistics on those populations (i.e. Best, Average and Worst) of each generation, known as Final Statistic File. In order to utilise both of these files to the fullest extend the application treats them separately.

Each of the two files are produced for either the Training Phase of the GP system, or Validation Phase. Table below illustrates the different possibilities:

Single Objective Mode (multiple and single run)	
Training phase	Population Statistic File
	Final Statistic File
Validation phase	Population Statistic File
	Final Statistic File

Table 2: Output files

4.3.3.1.1 Training

The training phase is the phase in which the generation constantly perfect each other. These generations will be testing themselves on a set of historical data and will produce result each time. These results will then go through a fitness test to eliminate a under performing individual. This will create a series of generation that is by generation-based graph.

As said, two files are produced by the GP in training mode.

The first file that contains the population of each generation (Population file) is a file that contains all the generation data of that specific run.
In this file, the Annualised Return, Annualised Standard Deviation and the Sharp Ratio of each Individual are displayed. Therefore, there would be a list of individuals with these three values.
But a problem encountered was that this file did not specify which of these individuals are for which generation. In order to make that clear, the GP system had to be modified in order for the application to understand its results. Having done so, these generations were then divided by a Generation Separator String that can be specified by the user in the application.

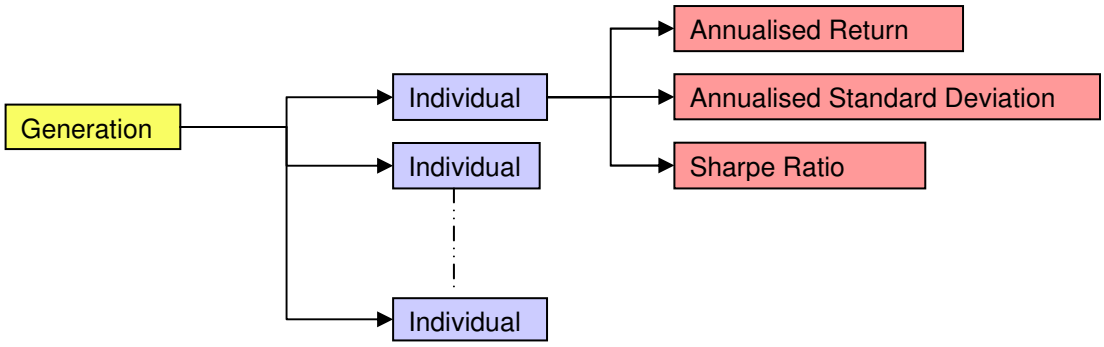

Figure 18: Training Mode - Individual Attributes

After the data is read, it will then be restructured into a completely new database that will be fully understandable for the graph section of the application. These changes take place inside a completely different class, which is responsible for making the conversation.

After the conversation, the user has the ability to view every generation's statistics.

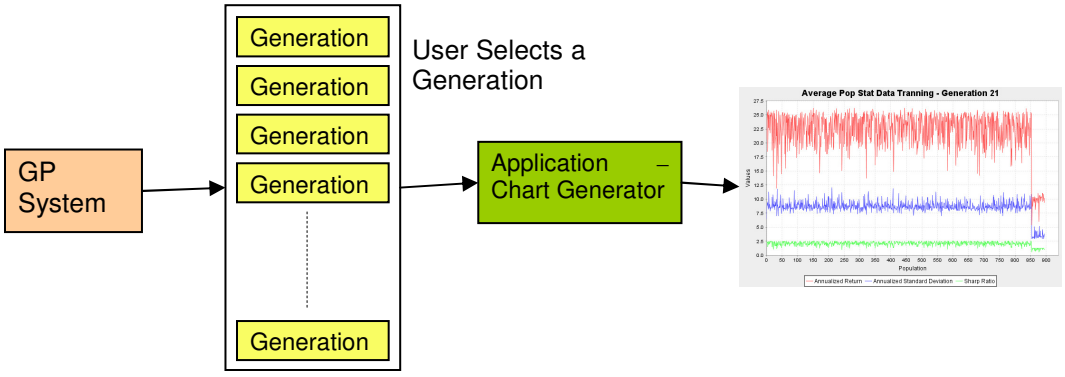

Figure 19: Training Mode Stages

The second file contains the statistic performed on these generations called Final Statistic file. These statistics are the Best, Worst and Average of each Generation. So there would be best, worst and average of annualised return, best worst average of annualised standard deviation and best, worst and average of Sharpe ratio.

Since there are nine different values for each generation, then a choice is given to the user for which of these graphs and different values they want to select. This will enable them to view the graph of their choice.

As a result, this comes in form of 6 option boxes where three are the Best, Worst and Average and the other three are Annualised Standard Deviation, Annualised Return and Sharpe Ratio.

4.3.3.1.2 Validation

The Validation phase is the phase that uses a small population of trained data to experiment on a new historical data. The Validation phase is used to assess the performance of the GP system and its accuracy on future predictions.

Validation Phase will also produces two files:

The first file would be the Population statistic file that contains the individuals of the selected population we acquired from the trained data. The number of individuals would not be significant. These individuals as before have their value of Annualised Return, Annualised Standard Deviation and Sharp Ratio with them. In this case, there is no generation. This is because as said, a set of population of trained data was chosen to experiment on a new historical data, and thus there will not be any generation involved. Since there are no generation involved, it will display the trend of each of these attributes of an individual in one sole graph.

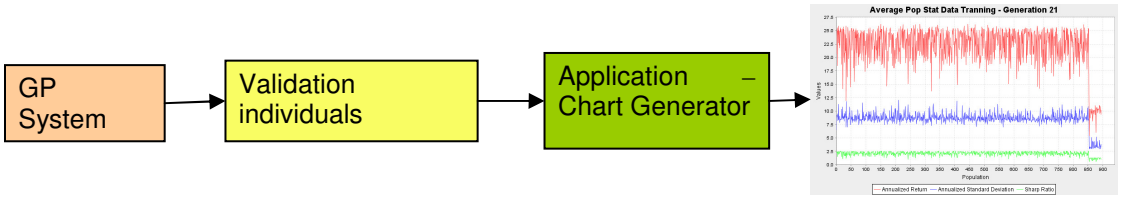

Figure 20: Validation Stage

In addition, the second file that contains the statistics for each generation is similar to what we have seen in the training mode. It will still be the statistics performed on each generation (Best, Worst and Average) on each of its value.

The Graphical Representation of this file would be same as trained data. The user is given a choice for which of these graphs and different values they want to select. This will enable them to view the graph of their choice.

4.3.3.1.3 Multiple Runs

The second part to this application is getting multiple runs of GP outputs and displaying them. As we discussed, the first part of the application was getting the output generated by a non-parallelised GP system, but on this section, it will be parallelised GP system.

There are several parts to this section. The first step is to get the statistical data from the parallelisation. As we know, the parallel system generates a series of folder in which the files are stored. Knowing that, the application uses a searching algorithm to dig through the folders and files and finds the specific item that is needed.

It should be mentioned that the filename could also be specified and edited by the user. This is mostly because each GP system can have different file names and path for their files and folders.

After the files are located by the searching algorithm, the application will then use a reader to read the files and store into a structural data. This structure will be constructed in a way to best fit the graphical section of the application.

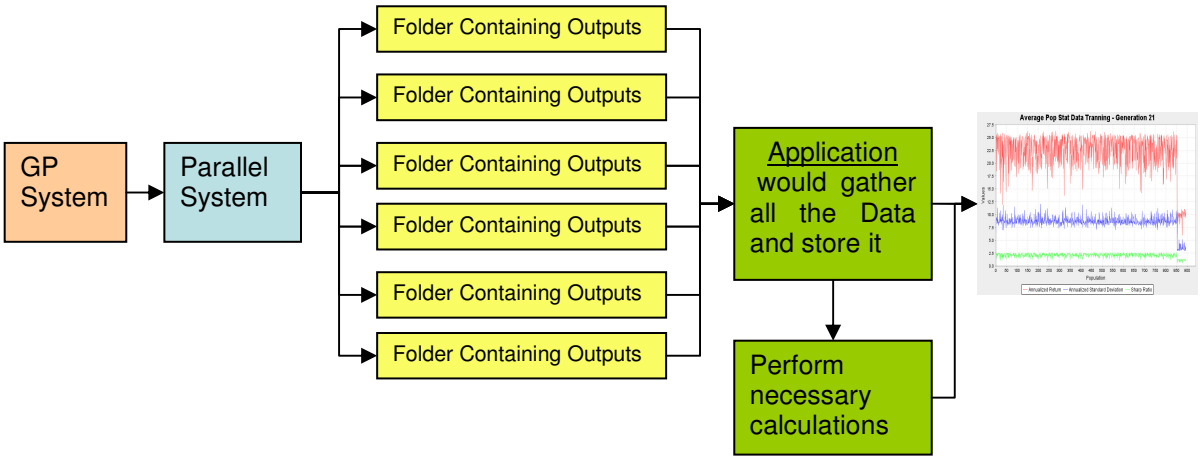

Figure 21: Multiple Runs Stages - From GP to Graph

Having located and retrieved the data based on search algorithm, the program will then try to combine all the data together and store it into a new structural data, which the mean of the data will then be calculated. After doing so, it will pass it on to the graphical representation of the application to display the necessary data. . Because of the nature of GP system, each run may produce different number of individuals and thus different outputs; therefore, this would be an issue when combining the data.

Since in both Validation and Training the data would be similar, therefore the application would have no problem retrieving it.

After being retrieved, the application would combine the data together. This would be tailored for each type of file, as their template is different.

After being combined, it will then go to the next level, calculate the mean of all the files, and store it into a new structured array.

Having done so, the new structured array will then go to the graphical section of the application and will be represented in graphs.

4.3.3.2 Part II: Multi Objective

The Second part of the application is the Multi Objective file. This would be very much different to the single Objective mode, as it will use several objectives to achieve its goal.

With regards to Single Objective, in Multi Objective each individual has a fitness value that is a vector of numbers. Therefore, the comparison cannot always be correct since the Fitness value is not a single integer.

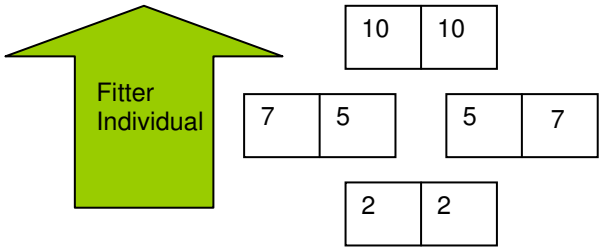

Figure 22: Individuals Fitness Value

As we see in the example (Figure 22), the first individual (10,10) is the fittest, while the less fittest is the last individual (2,2). Since these individuals contain an array of vectors for fitness value, we cannot always compare them together and give a definite answer. For example, as it is seen in the second and third individual, it would be impossible to say which one of them are fitter according to only element.

In the application, the Multi objective has a very similar role to the Single objective. Its files are somehow similar and there would be a resemblance between the two. The detail is in Table .

Multi Objective Mode (multiple and single run)	
Training phase	Population Statistic File
	Final Statistic File
	Final Front Values File
Validation phase	Population Statistic File

Table 3: Multi Objective File output

4.3.3.2.1 Training Run

In multi objective training mode the GP would produce three type of files:

The first file would contain the population of each generation (Population file). It would hold three different values for each individual within the population. These values would be the individual's annualised return, its standard deviation and the sharp ratio.
This file would be similar to the file we have seen in the single objective mode, but the difference would be in the number of individuals in each generation, since it is multi objective.

The first file that contains the population of each generation (Population file) is a file that contains all the generation data of that specific run.
In this file, the Annualised Return, Annualised Standard Deviation and the Sharp Ratio of each Individual are displayed. Therefore, there would be a list of individuals with these three values. But the problem was that this file did not specify which of these individuals are for which generation. In order to make that clear, the GP system had to be modified in order for the application to understand its results. Having done so, these generations were then divided by a Generation Separator String that can be specified by the user in the application.

4.3.3.2.2 Validation Run

In the validation phase, again a selected number of trained data is put to test by giving a new historical data. This will result in producing one file with the individual's performance. This would be same as population data but there will not be any string separators as there are no generations involved.

These selected individuals will again each have three different values. The values are Annualised Return, Annualised Standard Deviation and Sharpe Ratio.
The application will receive these file and display them.

4.3.3.2.3 Multiple Runs

In the multiple run also, the front-end application uses a searching mechanism by looking through a user-browsed folder and identifying the relevant files.

One of the positive attribute of this application is that it will automatically identify whether the file found was in the training mode or the validation mode.

Consequently, the application will combine the result found within the folders and display them accordingly. It should be mentioned that the process would be the same.

5 Testing

5.1 Testing

In this project, we deal with three stages. They are the GP system modification, the implementation of GP on parallel computers and the font-end application. The latter stage, which creates a program in java environment, is testable.

It is essential for the program to produce the correct results; this is because of the sensitivity of the data produced. The trends as well as the values are regarded as highly important. Unexpected results could arise because either the data is wrongly structured or the GUI is miscommunication with the application.

Since this project is not algorithm based and would not be changing any code, the testing would not be in terms of any classes. Hence, acceptance test will appear. This test carried out manually.

One of the tests that were carried out was the comparison between the graphs that the spreadsheet produces vs. the graph produced by the application. This can be very constructive, as it will spot any changes.

In addition random specific data from the GUI graphs in each section where picked and compared to the corresponding value in the CVS files. In graphs where calculation on data is carried out then displayed, (ex. averaging), similar calculation is carried by hand on the specific data.

6 Validation

In this section, we will be evaluating what this project has achieved. To do so, we will first explain the modifications on the GP System and then how the Runscript program enables the GP to function on the parallel computers. We will then move on to how the Java based program gathers data from the parallel computer and displays it. Also on the Java program, we will be walking through the different output data produced by GP (Single Objective/Multi Objective) and the different Modes we can choose (Single Run/Multiple Run).

6.1 GP Experiment

First, we looked at the GP modification that primarily focused on the output generated by the GP system. It diversified each output and diverted it into separate directories. This prevented the overwriting of the output data. Since a unique directory is created every time the program runs, a unique distinctive directory is produced.

Second, we looked at how the Runscript connects GP system to the parallel system. This connection is from a .SH file, which directs the parallel system to the GP class. In the Runscript we can also specify how many runs we would like it to have.

Third, we looked through how the GP system will be running on parallel system and how the program will evaluate and display the data.

To start, we will be examining the four possibilities of the GP system.

The **foremost** possibility is Single Objective Training Mode. We would be making sure that the GP system and classes stored in 'Keter' machine is running on Single Objective Training mode. This can be done by looking at the class InvestSim.java. If the InvestSim class needed some modification, we would have to update the class in 'Keter' and compile the java file.

Having done so, we examined the Runscript program on 'Keter' and ran multiple jobs on it. This was done by submitting the Runscript file to 'Keter' and doing so we waited until the computation is finished and the Runscript is executed. It will then output the

data file produced by the GP system and stored on separate folders every time executed.
After that, we gather the data and feed it into the program, which will then be reading the output data and structure it into a table (ArrayList). Then we converted it into a class recognised by the graph representation (XY Series Class). The process of conversion should be done for every single data that is fed into the program. After the conversion is done, the data will be displayed almost instantly. This would be in the form of an interactive graphical representation where the user can zoom-in zoom-out and save the graph as a PNG file.

Here is the screen shot of the system.

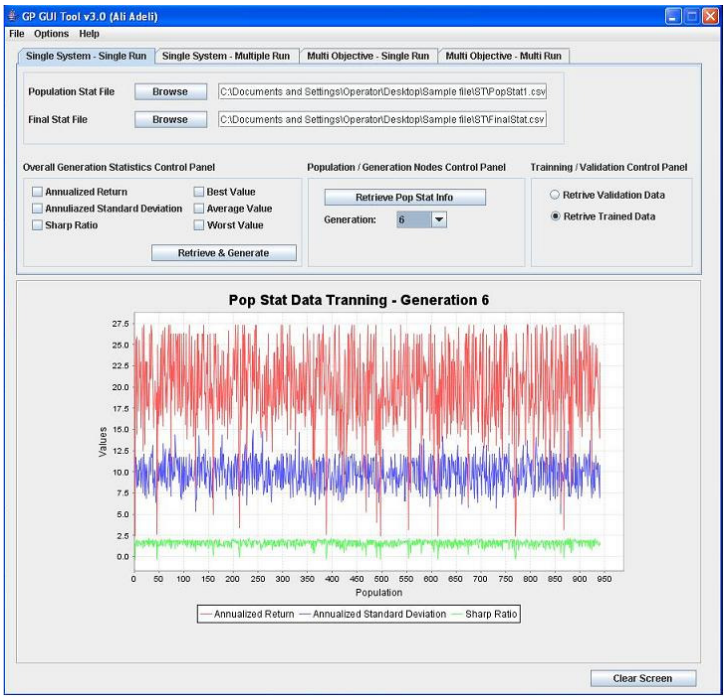

Figure 23: Screen Shot - Single Objective Training

The **second** possibility is Single Objective Validation Mode. Again, we made sure that the GP system and classes stored in 'Keter' machine is running on Single Objective Validation mode. This can also be done by looking at the Class, which is responsible for such changes, InvestSim.java. If the class was not initialised, we then had to retrieve the class from Keter, make the necessary modification to the class, upload the class and recompile it. After doing so, we gathered the data and again feed it into the front-end program. The program will then read the files, converts it into a recognisable class understood by the graph producer, jfreechart. After that, the process would be same as previous case.

It should be mentioned that both of the Single Objective Training and Validation mode have similar output data. They both have Population File, which contains the individuals. These individuals are produced in every generation in Training mode, and are selected on Validation mode. The second file that is similar is the Final Statistic File, contains the statistics performed on each generation. These statistics are the average, best and worst of each individual within each generation.

Now we will be looking at an example within Single Objective Validation for Single Run and Multiple Run.

This is an example of a training GP run. This is where the generation in GP system consistently perfect each other by testing on an historical data and going through a Fitness test. To expand further each generation contains individuals, and each individual represents a trading strategy for a portfolio.

We can see in the graph below that the average annualised return was sharply increasing while annualised Standard deviation had a slight increase. This graph below is for a single run on a single machine.

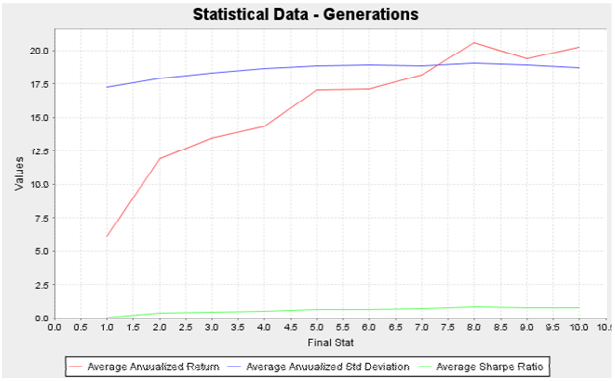

Figure 24: Single Objective - Single Run

The graph above (Figure24) represented a single run while the graph below will give an outline of a parallelised run on the 'Keter' System. These will be made of several runs, in this case 4 runs.

We can see on the graph below (Figure 25) that the annualised return has much more increase in the first five generation and remained constant through the next five while on the previous graph there was more or less a steady increase through out the generations.
In addition, we can observe that the annualised standard deviation is also increasing in the first 5-generation and decreases as it goes on while on the previous graph, its someway rising gradually. However, on both cases, the Sharpe ratio is constantly increasing over 10 generation.

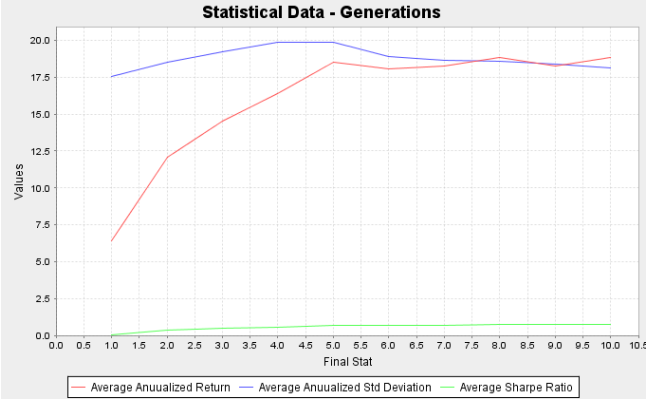

Figure 25: Single Objective - Multiple Run

The graphs we have just seen was the GP operating in single Objective mode. We will now look at Multi objective and how the GP will operate and their difference between Single Objective and Multi Objective.

The **Third** possibility is Multiple Objective Training Mode. In this section, each individual holds a vector of integers representing their fitness value. We first made sure that the GP system and classes stored in 'Keter' machines are running on Multi Objective Training Mode. This can also be done by looking at the class that is responsible for such changes, InvestSim.java. Yet again, if the class was not initialised we must modify and make necessary changes to it. The parameter file is also changed. After doing so, the data was then produced. The program will feed in this data and display it. The process would be fairly the same. A screen shot of this section can be seen in Figure 26.

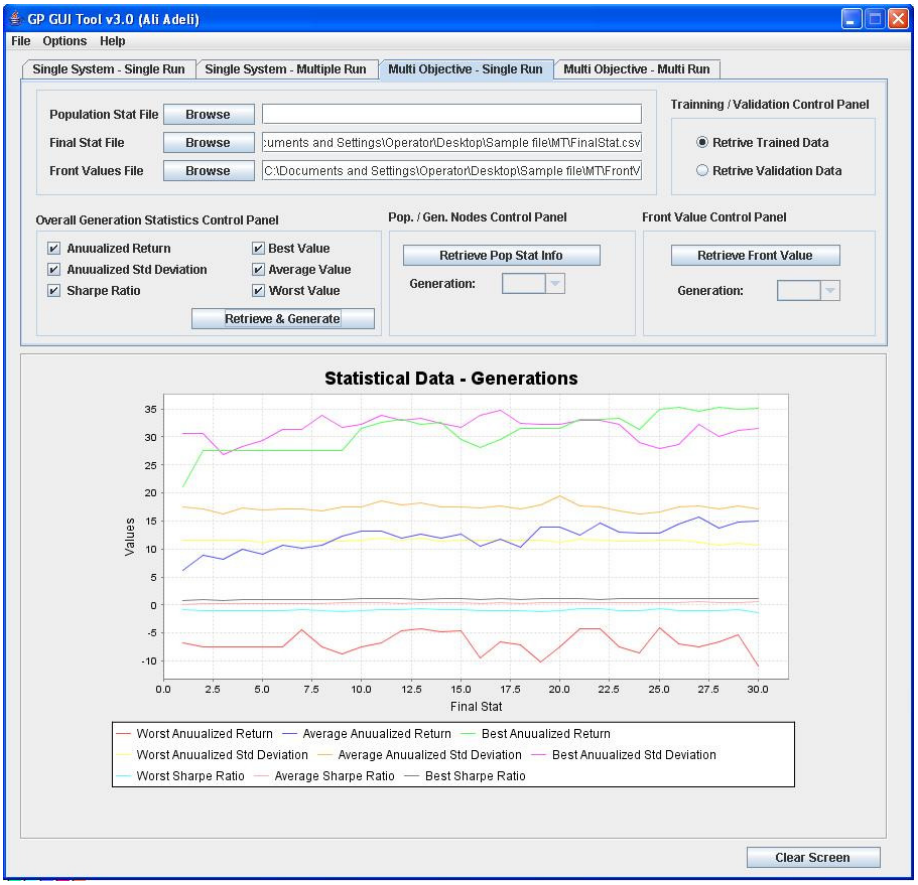

Figure 26: Screen Shot - Multi Objective Training Mode

The **fourth** and last possibility is Multiple Objective Validation Mode. This section would fairly be same as the last section except that the modification done to the GP should be more than just changing InvestSim.java. We will also have to make changes to the parameter file as well is changing the file name of FinalFront.csv to validate-Front.stat . These changes are the requirement for the GP system to function.

We will now look at an Example of Multi Objective System Training mode.

The graph in Figure 27 shows a statistical data of a Multi Objective run over 30 generations. As we can clearly see, the Average Standard Deviation is a very high number comparing to the Annual Return. Nevertheless, this is as high as it was in the Single Objective Mode, fluctuating from 15 to 20, comparing it with Figure 24.

The important thing is that the Annual Return has decreased comparing to Single Objective. As we can see, regardless of One Run or Several Run, in the Multi Objective Mode, the Annual return will have to reach 12 at its highest to reach its peak while on Single Objective this value will reach 20. In Addition, when comparing the Sharpe Ratio, we will also find out that the Single Objective will be higher than Multi Objective will.

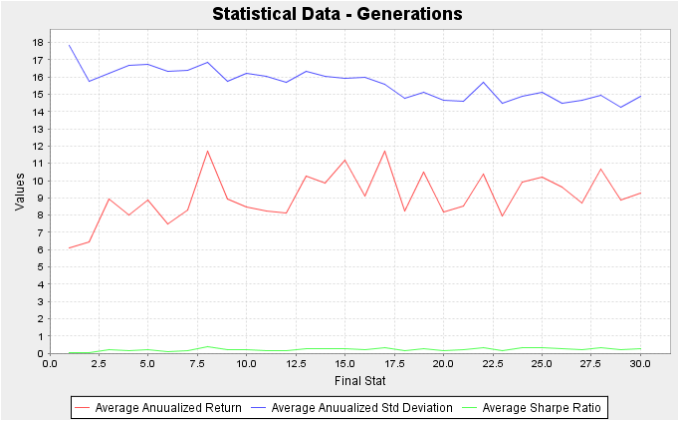

Figure 27: Multiple Objective - Single Run

Consequently, this will clearly demonstrate that in terms of trading strategy, it will be best to choose the single Objective mode, as it will give more return.

For Multi Objective Validation mode, this is where the GP system, having finished the training, will gather a selected number of individuals, which were created in the training mode, and uses their trading strategy on a new historical data. This will indeed show how the GP will operate having done the training.

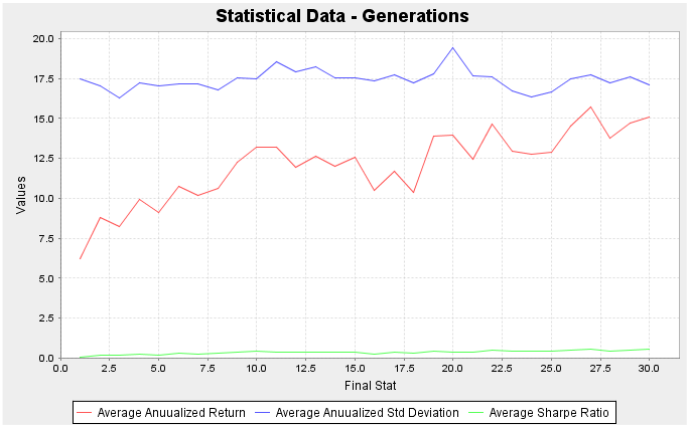

Figure 28: Multiple Objective - Single Run

For Example, we can see in Figure 28 that just as expected, the Average Standard Deviation is consonantly high compared to other values, but in surprise we see the

Average Annual Return exceeding our expectation and considerably rising. We can clearly see the difference between Multi Objective – Single run in Training mode and in Validation mode here.

6.2 Assessment

In reality, when the user is observing the graphs, they will eventually look at the performance of the GP system and how it will be ranked. The values are as important as the trend is. Ultimately, the user wants the GP to make profit. If the Return is high enough and the GP's Sharpe Ratio, which is a measure of assessing the GP, is good enough, than the GP would be highly considered.

Parallelisation of the GP will generate more output, and thus a broader range of data and consequently, a more accurate result.

7 Summary and Conclusion

In this final chapter, the project is evaluated as a whole in terms of what has been achieved, the challenges that was faced and how they were overcome is discussed. Furthermore, the project was evaluated in terms of what could have been achieved given more time.

7.1 Summary

In this project the objectives stated in section 1.4 were met. They are as follows

- A modification was done to the GP system to allow it to function on parallel computers. This modification will not only diversify the output into several folders in 'Keter' machines.
- A UNIX based program was designed and implemented to allow the GP system to work on the 'Keter' machines. This program is the bridge between the GP and 'Keter' system and controls concurrent GP system.
- A java-based program was designed and implemented to receive the output of the parallel computers and provide automatic interactive graphical representation.
- The program not only can view parallelised data, but also it can view single data of the GP system, not parallelised, regardless of Single Objective or Multi Objective.
- The program contains a graphical user interface for the user to interact with.
- The implementation as a whole is flexible enough to contain further extensions.
- Further to the project, A step-by-step documentation for modification of any GP system along with how to get 'Keter' account and how to set the Runscript in 'Keter' System was produced. This is aimed at current and future researcher who would want to parallelise their algorithm. Parts of the report are also useful for researchers who might not be working on GP systems.

7.2 Challenges

There were many challenges faced in this project. This was namely due to unfamiliarity to 'Keter' parallel system, as well as lack of documentation regarding the 'Keter' and in general UCL parallel machines. However, by investigating thoroughly and learning about the concept of parallel computers, I overcame those challenges.

In addition, another challenge I faced was unfamiliarity with researchers GP system and how to modify it so that it will be suitable for parallel computers. This was indeed challenging because of time-constraints I had to develop a method to modify the GP system to work on 'Keter'. It involved thoroughly going through different parts of the code, understanding it and then modifying it.

Lastly, my final challenge was to design and implement a user-friendly program to use the data derived from the parallel computer and automatically display a graphical representation of it. This program reads the CSV files and to structure them into a separate database and display them in an interactive graphical representation.

Because of the unique mixture of these technologies, there were many challenges faced. I overcame these challenges by breaking the project into three sections, setting goals and small targets for each of the sections and then executing them. This proved to be very successful and crucial in completing the project on time.

7.3 Further Work

One of the first extensions that can be added to the front-end program is the ability to compare two systems together. It would be very appealing for two researchers to evaluate their data after being parallelised in the 'Keter' system. Their data can be measured by performing a ranked T-test that is ideal for comparison.

References

[1] Eckart Zitzler, Marco Laumanns, and Stefan Bleuler "*A Tutorial on Evolutionary Multiobjective Optimisation*"- Swiss Federal Institute of Technology (ETH) Zurich

[2] Bob Fulks *"Sharpe Ratio",* MIT Press*, June 22 1998*

[3] Felix Streichert "*Introduction to Evolutionary Algorithm"* – University of Tuebingen, April 2-4 Conference 2002

[4] Carlos A. Coelho Coello *"A short Tutorial on Evolutionary Multiobjective Optimisation*" Department of Ingenieria Electriaca, Instituto Poltecnico, Mexico

[5] William F. Sharpe "*The Sharpe Ratio"* – Stanford University - Reprinted from The Journal of Portfolio Management, Fall 1994

[6] UCL Research Computing Group.
http://www.ucl.ac.uk/research-computing/index.php

[7] Eclipse Project website
http://www.eclipse.org/

[8] Jigloo SWT/Swing GUI Builder for Eclipse and WebSphere
http://www. cloudgarden.com/jigloo/

[9] JFreeChart Project
http://www.jfree.org/jfreechart/

[10] Wei Yan and Christopher Clack, "Behavioural GP diversity for dynamic environments: an application in hedge fund investment", GECCO 2006

[11] Wei Yan and Christopher Clack, "Diverse committees vote for dependable profits", GECCO 2007

[12] Wei Yan and Christopher Clack, "Evolving robust GP solutions for hedge fund stock selection in emerging markets", GECCO 2007

[13] Suneer Patel and Christopher Clack, "ALPS evaluation in financial portfolio optimisation", IEEE Congress on Evolutionary Computation 2007

[14] Ghada Hassan and Christopher Clack, "Multiobjective Robustness for portfolio optimization in volatile environments", UCL 2007

8 Appendices

Appendix 1: User Manual

Please find attach Document.

Appendix 2: Code Listing

Please find the CD.