Final Report
Genetic Programming
Project

Author: Vishal Yelisetti, Ou Li

Creation Date: 3/21/2015

Last Revised: 5/03/2015

Version: 1.0

Problem Description

Background Information/Available Alternatives

Genetic programming is a model of programming which uses the ideas (and some of the terminology) of biological evolution to handle a complex problem. Of a number of possible programs (usually small program functions within a larger application), the most effective programs survive and compete or cross-breed with other programs to continually approach closer to the needed solution. Genetic programming is an approach that seems most appropriate with problems in which there are a large number of fluctuating variables such as those related to artificial intelligence.

Problem Description

The purpose of generic programming project is to design and implement a system which can select a function which close to a target function within a time period.

Requirements Analysis

Client

I. Client Requirement

1. Build Generic Programming System.
2. Target Function $\frac{2x^2-3}{3}$
3. 15 minute limitation.

II. User Interface

A Graphical User Interface can run the simulation and a final report displayed when system is finished.

III. Target Client(End user)

The clients want to find a closet matching function from a target function.

Data

I. Data Models

Terminal set: The terminal set is composed of the inputs to the Genetic program. [0..9], Y, X.

II. Data Collecting

1. Training Data Size: 70-100
2. Range: -15 to +15

3. Accuracy: 5 decimal Places
4. Calculated and save to a text file

III. Setting Collecting

1. Maximum height of initial tree
2. Population Size
3. Crossover probability
4. Mutation probability
5. Crossover selection percentage
6. Mutation selection percentage
7. Fitness margin
8. Population Size
9. Max Execution Time
10. Initial Tree Height
11. Number of Crossover

IV. Function set

The function set is composed of the statement, operators, and function available to the Genetic program. +,-*,/, SQRT()

1. Fitness Evaluation: Takes the training data and initial tree and output the list of trees of their fitness values.
2. Determine termination condition: Takes single tree and fitness criteria settings and output a Boolean result.
3. Fitness function: compares produced output with expected outputs
4. Crossover: is a single point crossover and take as input selection of next generation population and crossover probability and output the crossover modified tree.
5. Mutation: takes the result tree from crossover function, mutation probability, the valid operator and new trees.

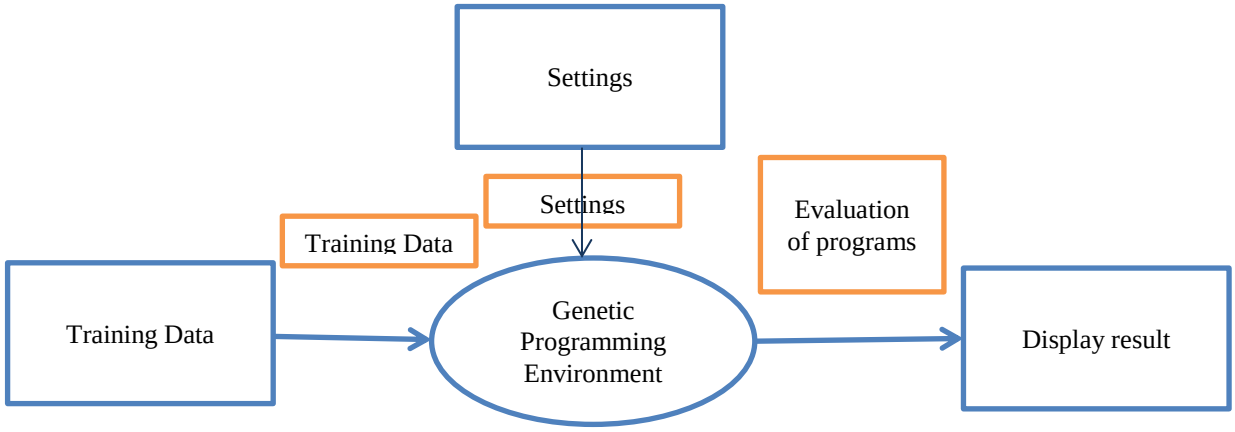
V. Reasons in selecting object-oriented approach

Consider the iterative methodology, ability to change requirement based on customer.

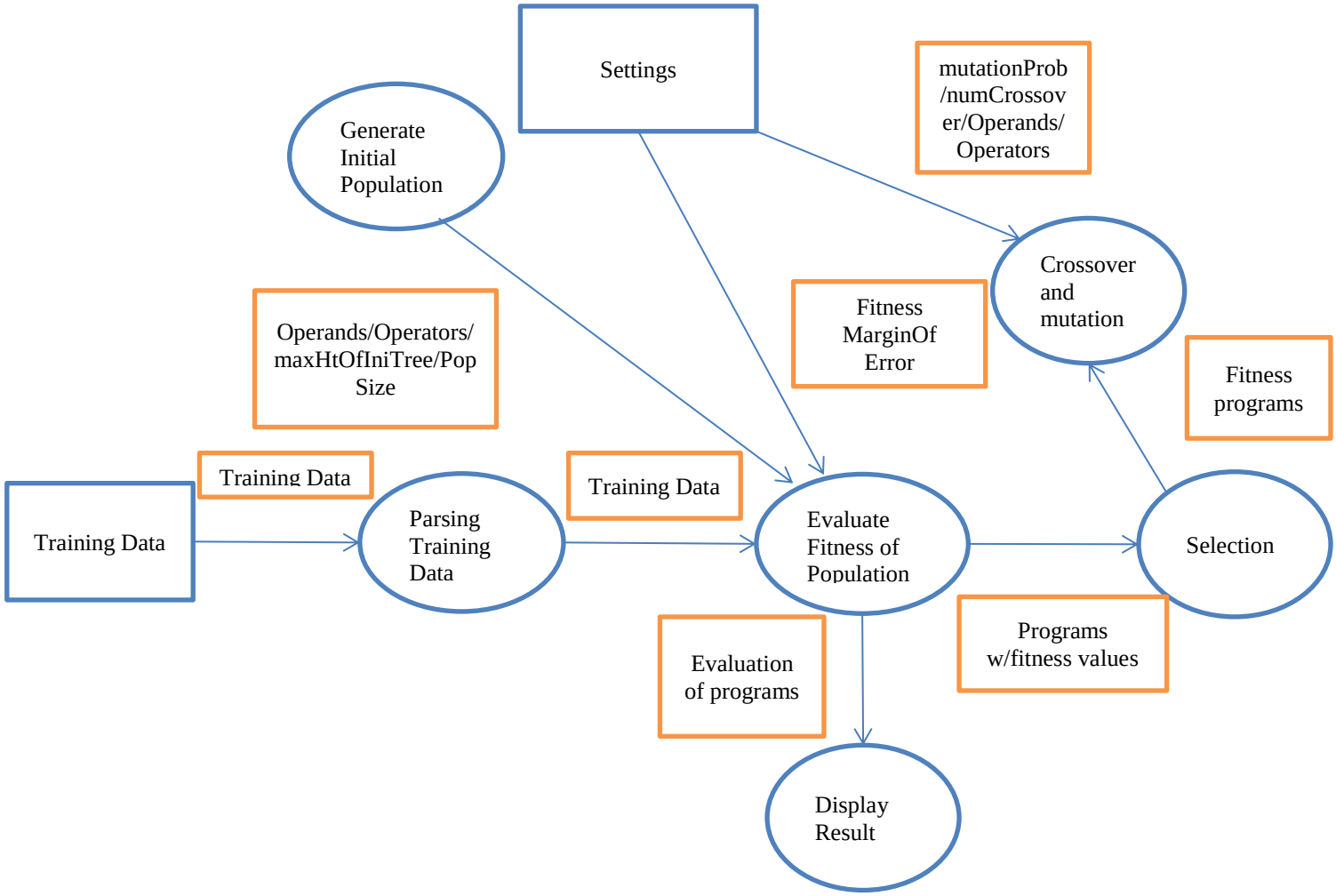
Function

I. Data Flow Diagram: Level 0

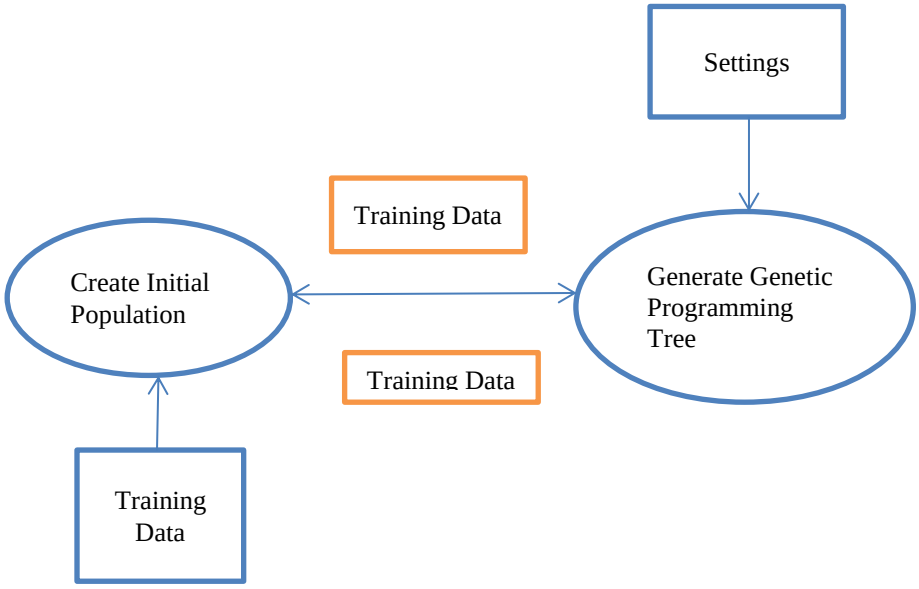
Data Flow diagram is to define all the process diagrams for the main process.



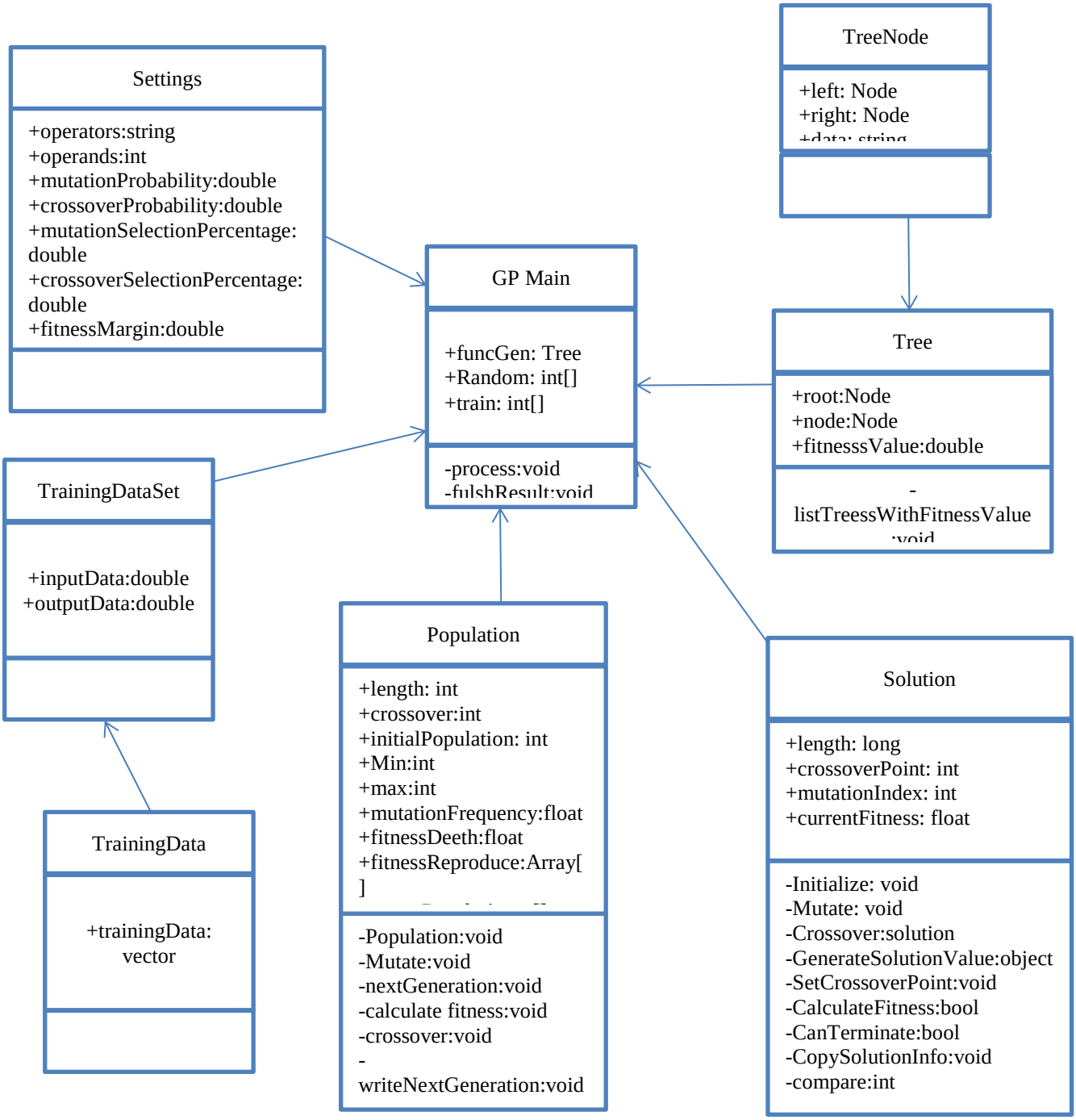
II. Data Flow Diagram: Level 1



III. Data Flow Diagram: Level 2



IV. Class Diagram/ Object Oriented Diagrams



- **Behavior**
  1. **States**

They are observable circumstances that characterize a system.

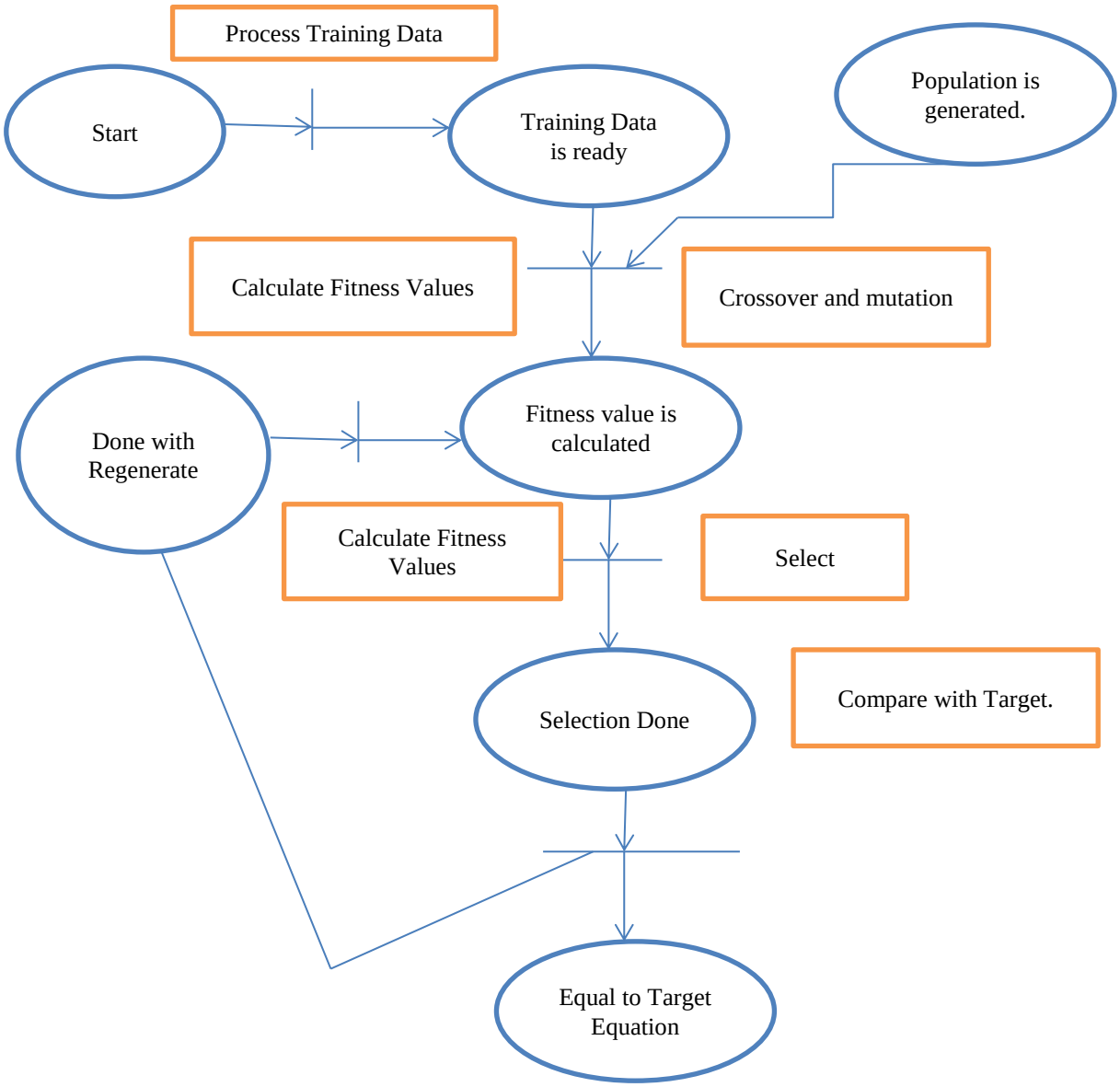
And possible states are Initial state, training data set is in the system, Fitness of population is evaluated.

2. **Events**

They are the events which cause a state transition.

And Possible Events are parse/read training data, generate high, generate trees, generate notes, calculate fitness values, Sort, Select, Mutation, operates.

Briefly graph shows in following.



.

System Design

I. System Architecture

The Genetic Programming System is developed using object oriented design principles in the Java programming language. Matlab was use an initial tool to gauge the appropriate functions and its common functions.

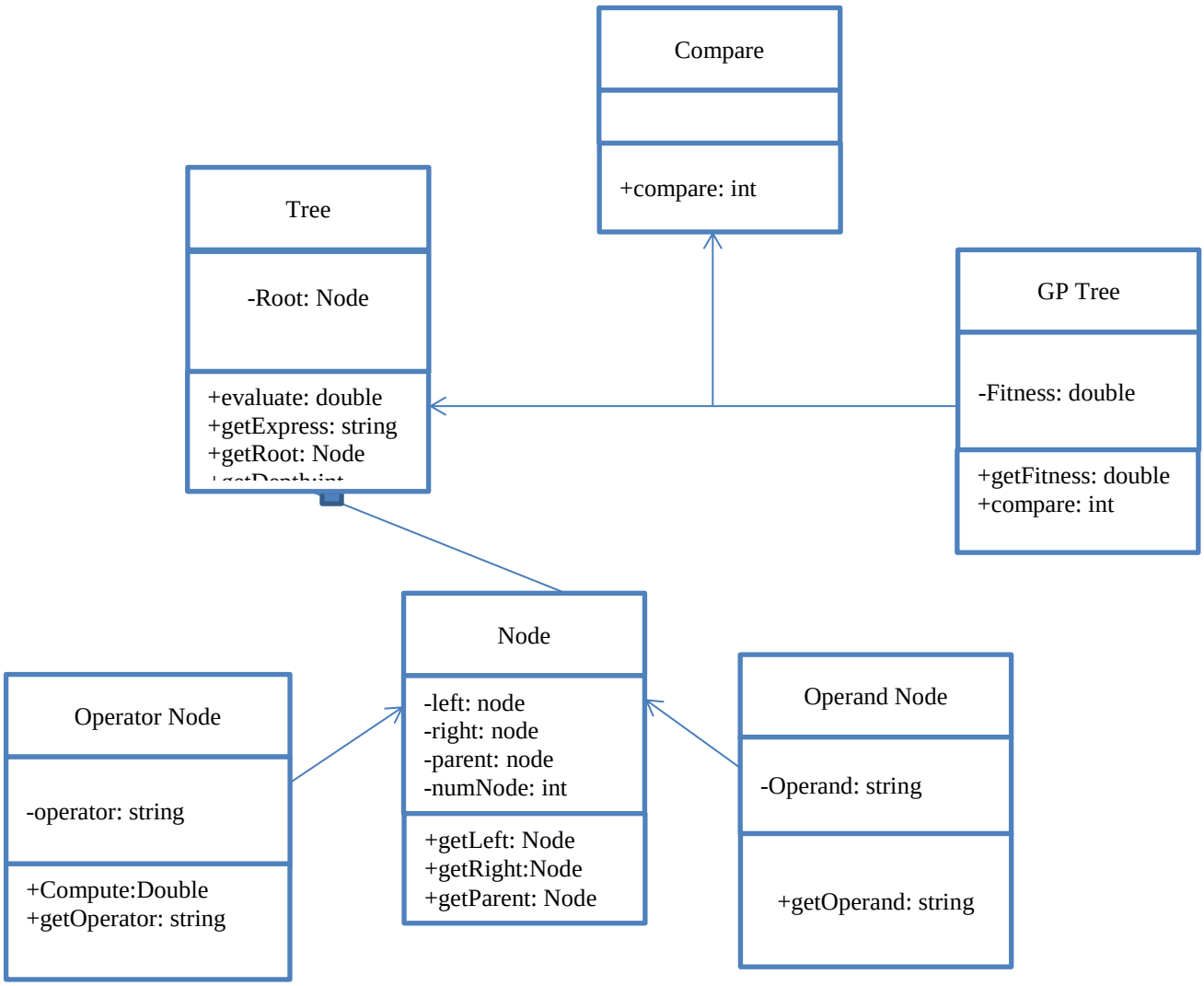
II. Design and Development

We were given the ability to create our GP system or use a pre-existing library. We chose to start from the ground and build the project on these fundamentals, as this would allow comprising of our own UI and giving us the flexibility needed to provide a sufficient answer. With the awareness that our project would change as the project drew to a close, we would have a better grasp of what needed to change in order to incorporate these fixes. The ability to change code on a pre-existing library would have been different and little more tedious.

Some key constraints that could arise in this project are as noted: developing an adaptive system that only works for our current system but any function. Other dependency is that, we were to be under the notion that our libraries would be created perfectly and gel together quite extensively. The other trade-off was that in developing from scratch, there is the notion that our competency with Java is high. Though, there is experience in the team, extensity will be tested nonetheless.

The choice of using an object oriented design and development approach using Java as our programming language was because of several reasons:

- The language has support for developing a graphical user interface. For our project however developing the graphical user interface was a low priority development task.
- Object oriented development is today a proven, mature, widespread, and successful approach in developing modern software systems. It offers a low risk, high re-use, well performing option for developing stable systems.



III. Path Chosen

We choose to use the binary tree data structure to represent a Genetic Programming Tree because of the greater flexibility and power of offered by such a structure in recursively traversing, performing genetic operations of crossover and mutation, and in the evaluation of the algebraic expression represented by each tree.

For representing a population of Genetic Programming trees, we used the Java ArrayList structure as it allow us to easily do the following:

- Iterate over the population and access it using indexes
- Convenient allow us to efficiently sort a population of trees

- Ability to travers through a large amount of population under a contain time.

Testing

Figure 8 – Overall Project Issues

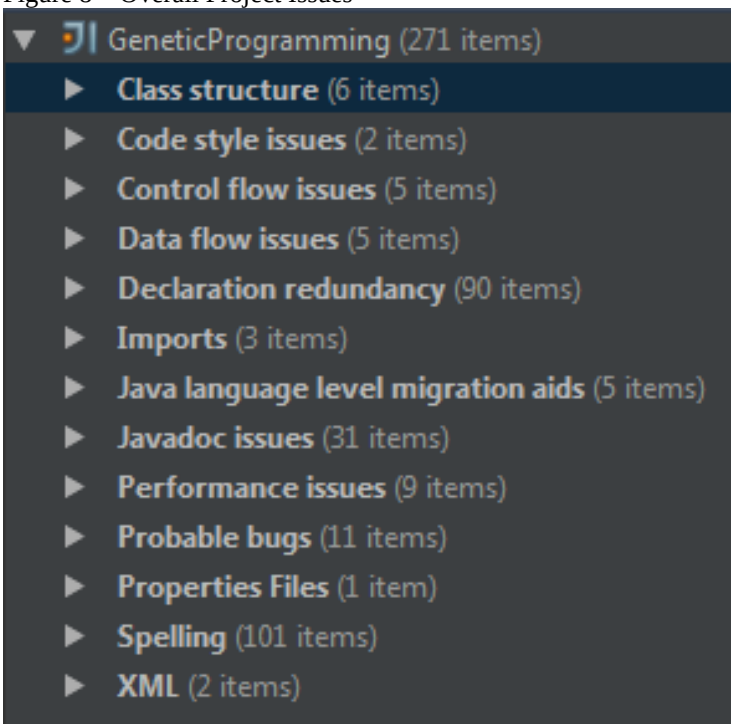


Figure 8 describes the overall issues compiled by the compiler. As per our knowledge, we can see that none of the bugs and issues are major. The largest issue or error occurs with spelling. This could be due to the methods not using consistent naming frameworks or misspelled words in the comments section of the code.

Test Cases

White Box - Sample

1. Name: **Sorting_Test**
   Expected Output: **Ascending order of trees in a sorted manner**
   Pass/Fail
2. Name: **Tree_Test**
   Expected Output: **Output an array size greater than 1**
   Pass/Fail

Black Box - Sample

1. Name: **Overall_time_completion**
   Expected Output: **System shall display the needed function in less than the "max-time" limit**
   Pass/Fail
2. Name: **Real-Time_functional_analysis**
   Expected Output: **System shall display the graphical user interface for the outputted result real-time**
   Pass/Fail

Post-project Analysis

Figure 1 – Overall Project Metrics

project	CCtot	CF	LOC	PF	TEST_RAT
project	793	10.21%	2450	41.18%	0.00%

Figure 1 shows the Overall Project Metrics. As one can see, the CC total value is not entirely accurate of the total Complexity of the project. This total is the total of all non-abstract methods displayed in the project.

The coupling factor is also at low levels, meaning that low numbers are ideal and easier to produce tests that test these artifacts. Consequently, it has been noted to be that it Is easily maintainable code.

Lastly, the PF implies the derivation of the classes from other classes. From Misra and Bhavsar, noting that with high PF factor, decreases the bug density as well as increases the quality of the code.

Figure 2 – CC metrics for troubled classes

class	OCavg	WMC
utilities.Utilities	3.11	10
utilities.GeneticOperators	3.08	13
Total		23
Average	3.10	11.50

Figure 2 and 3 describes the overall troubled classes overall complexity with respect to the system. This metric reports the average cyclomatic complexity of the non-abstract methods in each class. Inherited methods are not counted for purposes of this metric. This value ranges from 0 – 3, thus any value above 3 is regarded as high.

Figure 3 and 4 shows the weighted Complexity of each method by its class. As we can see, not a single method is above the harmful level of 50 and over. This shows that the written code is highly maintainable and highly sustainable.

Figure 3: Overall Cyclomatic Complexity of methods

class	△ OCavg	WMC
utilities.Utilities	3.11	10
utilities.GeneticOperators	3.08	13
utilities.TreePrinter	3.00	6
data.Node	2.42	20
data.Tree	2.40	20
data.GPTree	2.36	11
GeneticProgrammingMain	2.33	12
utilities.NodeFactory	2.20	16
test.GPUtilities	2.00	4
test.TreeTest	2.00	24
data.OperatorNode	1.88	8
data.TrainingData	1.62	8
test.SettingsTest	1.50	3
utilities.Settings	1.42	19
data.OutputData	1.38	22
test.GeneticOperatorsTest	1.33	4
data.OperandNode	1.25	8
utilities.BPTreeMain	1.00	4
test.TrainingDataTest	1.00	1
utilities.Node	1.00	1
test.NodeTest	1.00	3
data.GeneticProgrammingT...		11
Total		228
Average	2.06	10.36

Figure 4- Werighted CC by class

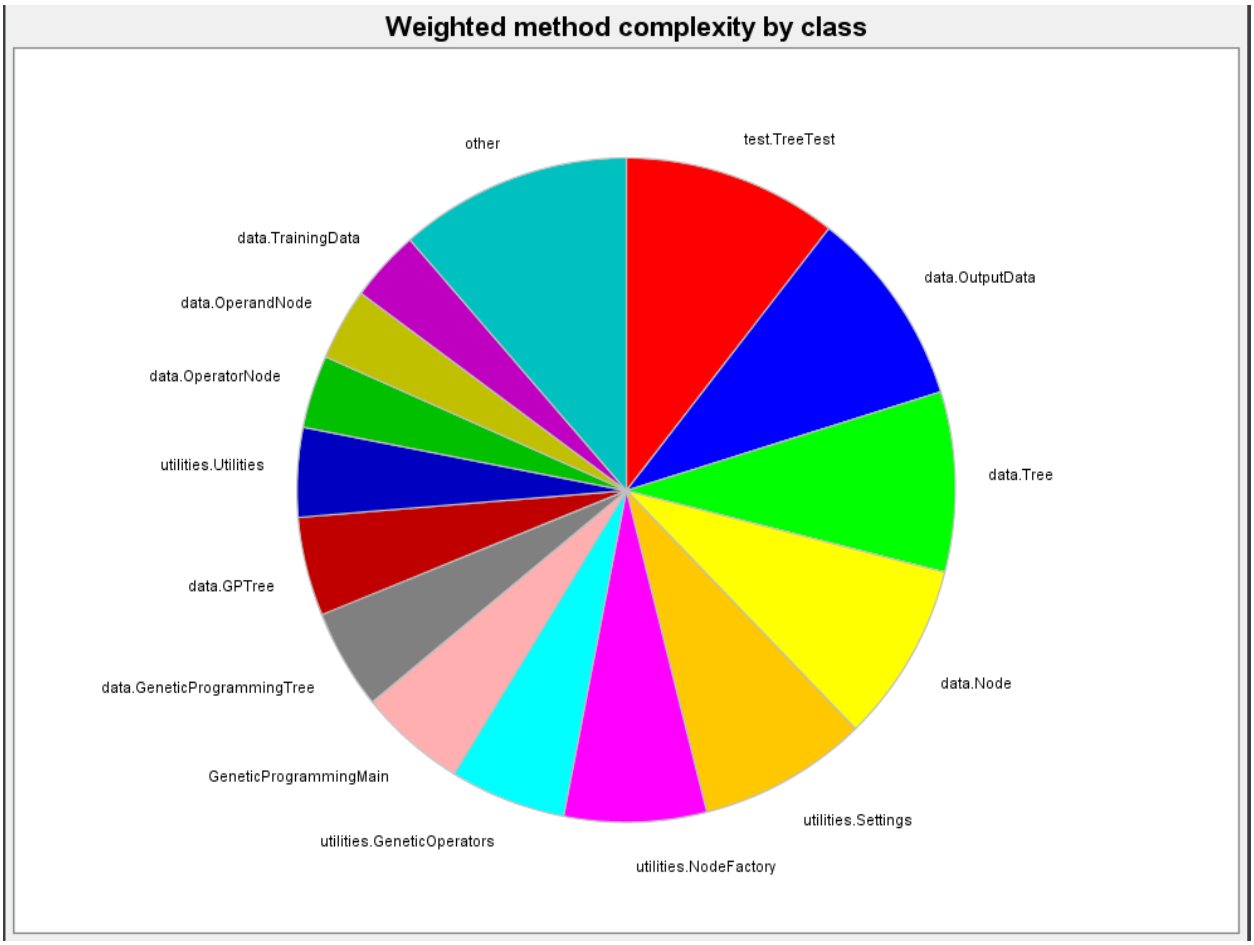


Figure 5 - LOC by package

package	LOC	LOC(rec)	LOCp	LOCp(rec)	LOCt	LOCt(rec)
data	987	987	987	987	0	0
utilities	803	803	803	803	0	0
test	480	480	480	480	0	0
	180	2450	180	2450	0	0
Total	2450		2450		0	
Average	612.50		612.50		0.00	

Figure 5 shows the overall Lines of code per package. There are several metrics here that are abundant; however, this metric clearly shows that no package is above the harmful level of 700 lines of code.

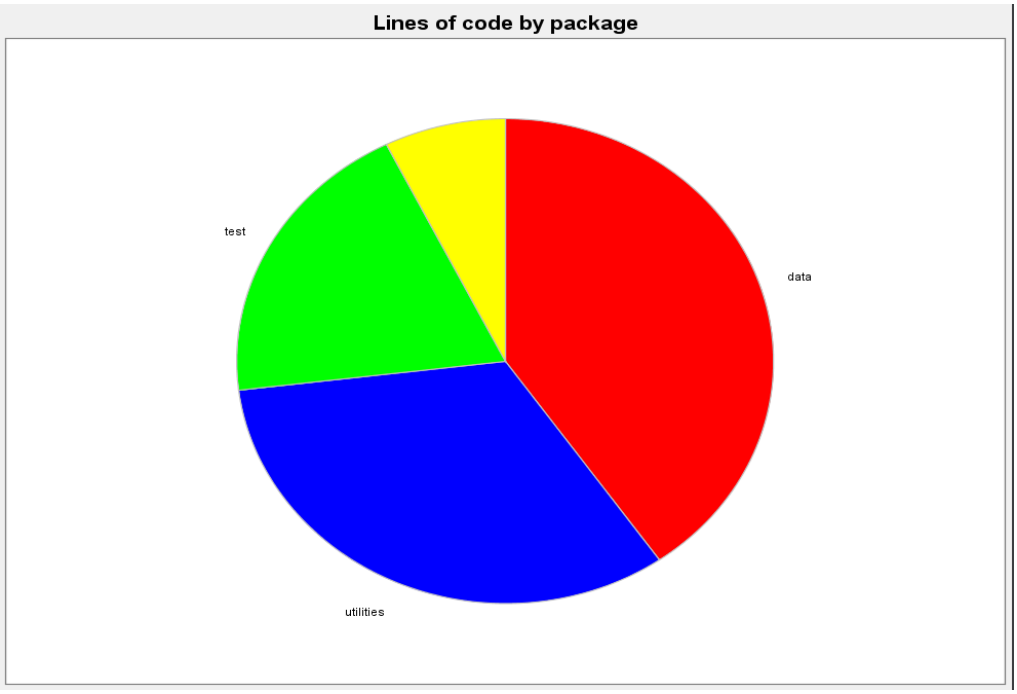


Figure 6 – Overall representation of the Lines of code in the project.. None of the lines of code are exceeding amounts that are known to be harmful. This is shown by the package. Each package consists of 5-6 classes, thus with 4 packages being shown, 22 classes in total, which averages to 111 lines of code. This is not saying that each class has these many lines, however, this is the best estimation of the overall subset.

Figure 7 – MOOD Metrics

project	AHF	AIF	CF	MHF	MIF	PF
project	49.91%	9.70%	10.21%	22.95%	24.38%	41.18%

Figure 7 describes the MOOD metrics that relate to each class. One metric that jumps out that is good to look at the in the method hiding factor. This metric desbies how the methods are being used in each class and by other subsets of classes. With a high value of the MHF, this would mean that all the methods are public and thus can be accessed and changed at any given time. Howeever, if all the methods are private, it would mean that each method cannot be accessed by othe classes and would need to be defined in the respective classes each time. Relating to our project, the 22.95% metric indicates that close to quarter of the methods are not hidden and can be easily accessed.

Lesson Learned

I. **Software Engineering Models**
Trading off software engineering models is a really good topic to discuss and implement in the real life.

II. **Testing Methods**
Testing is an important phase in software developing life cycle. By implementing what we learned in class, we have a good understanding about the concepts.

III. **Measurements**
All our teammates are currently software engineers. But rarely think about measurements of the project. It's really a good topic to extent.

IV. **Time Management**
Balancing time is important. Especially for the finally time, things are all get together.

V. **Documentation**
Don't postpone the documentation. Documentation should go along with the project. Postponing the documentation might cause lose design details.

Project Summary

I. Design and Implementation:

1. Software development life cycle
   Following the software development life cycle, mainly includes Requirement Analysis, Implementation, Testing, Post-Project Analysis.
2. Object-oriented approach
   Divided up into stages going from abstract descriptions of the problem to designs then to code and testing and finally to deployment. The earliest stages of this process are analysis and design and the analysis phase is also including requirement acquisition. And also, the implementation is mainly focus on the extension and maintenance because of knowing requirement changes will happened later time.
3. Data Structure
   This project involves mainly three types of data, Trees and Node, Training set and settings data.

II. Tools

1. GitHub: Tracing the implementation details.
2. IntelliJ: Implementation IDE.
3. Google Doc: Report group work and discussion.
4. Skype: Web meeting.
5. Microsoft Office: Finalize project plan and final report and presentation power point.

Weekly SCM Files and Folders

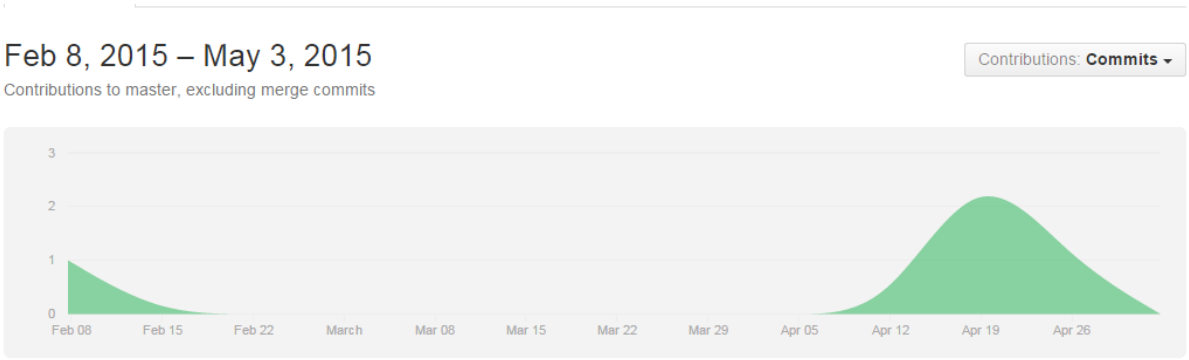
- **SCM Template**
  1. **Name of work**
  2. **Revision Date**
  3. **Changes**
  4. **How**
  5. **Where**

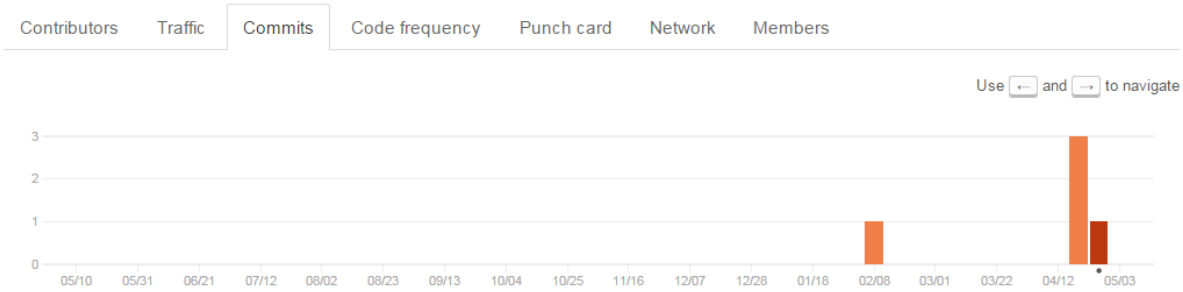
SCM Template					
Week 2					
Name of Work	**Revision Date**	**Changes**	**How?**	**Where?**	**Who?**
Problem Description	2/21/15	Draft	Word Document	Google Document	Ou
Initial Graphs	2/26/15	Graphical Representation of target functions	Matlab	Matlab	Vishal
GitHub Repository	2/27/15	Draft	GitHub	Online	Ou/Vishal
Week 3					
Intial Release to Google Docs	3/4/15	Matlab Code and Explanation	Matlab	Online	Vishal
Week 4					
Requirement Analysis	3/6/15	Initial Draft of the Requirement Analysis	Google Doc	Online	Ou
Behavior vs Development	3/10/15	Initial draft of Require Behavior and Development	Google Docs	Online	Ou/Vishal
Week 5					
UML	3/14/15	Initial Design: No nodes	Google Docs/ Paint	Online	Ou
UML	3/20/15	Initial Add-in of nodes/functions and classes	Google Docs	Online	Ou
OO Diagram	3/21/15	Initial Start at OO Diagram	Google Docs	Online	Vishal
Week 6					
	3/14/15	Data structure- Part1	IntelliJ	In person	Ou/V

			and words		ishal
	3/20/15	Data structure-Part2	IntelliJ	Online	Ou
	3/21/15	Reproduction –Part1	IntelliJ and words	In person	Ou/V ishal
Week 7					
	3/22/15	Reproduction –Part2	IntelliJ	Online	Ou
	3/20/15	Reproduction –Part3	IntelliJ	Online	Visu al
	3/28/15	Reproduction-Part 4	IntelliJ	Online	Visu al
Week 8					
	3/14/15	Main	IntelliJ	In person	Ou/V ishal
	3/20/15	Testing	IntelliJ	In person	Ou
	3/21/15	Testing	IntelliJ	In person	Visu al
Week 9					
	4/4/15	Selection-Part1	IntelliJ	In person	Ou/V ishal
Week 10					
	4/11/15	Selection-Part2	IntelliJ	Online	Visu al
Week 11					
	4/13/15	Testing	IntelliJ	In person	Ou
	4/16/15	Final Report-Part1	Words	In person	Ou/V ishal
	4/19/15	Final Report- Part2	Words	Online	Ou
Week 12					
	4/26/15	Finalize Final Report	Words	In person	Ou/V ishal
Week 13					
	5/3/15	Presentation prepare	PPT	PPT	Ou/V ishal

GitHub Graphs

Commits





Work Plan

Breakdown Among Members

- Part 1 Input (Ou) 20 hours

  Population Size, training set choose

- Part 2 Operation (Vishal, Ou) [Note: this part is complicated, we will discuss about the plan together] 200 hours

  Trees crossover mutation

- Part 3 Display (Vishal) 30 hours

  Implementing a GUI to display / output to a textfile and analysis through Excel or so.

- Part 4 Post-Project Analysis 8 hours

Milestones

Milestone	Estimated Completion Date

Phase I: Requirement Analysis---------- 2/15/2015

Phase II: Design---------- 3/08/2015

Phase III: Implementation----------4/14/2015

Phase IV: Testing---------- 4/25/2015

Phase IV: Final Report Review---------- 5/03/2015

Phase IIV: Presenting/Release---------- 5/9/2015

User Manual

1. Requirement

JRE(Java Runtime Engine) 1.6

2. Installation

download release zip file from flowing link:
https://github.com/li1530/GeneticProgramming.git

3. Run

a.Through command line:

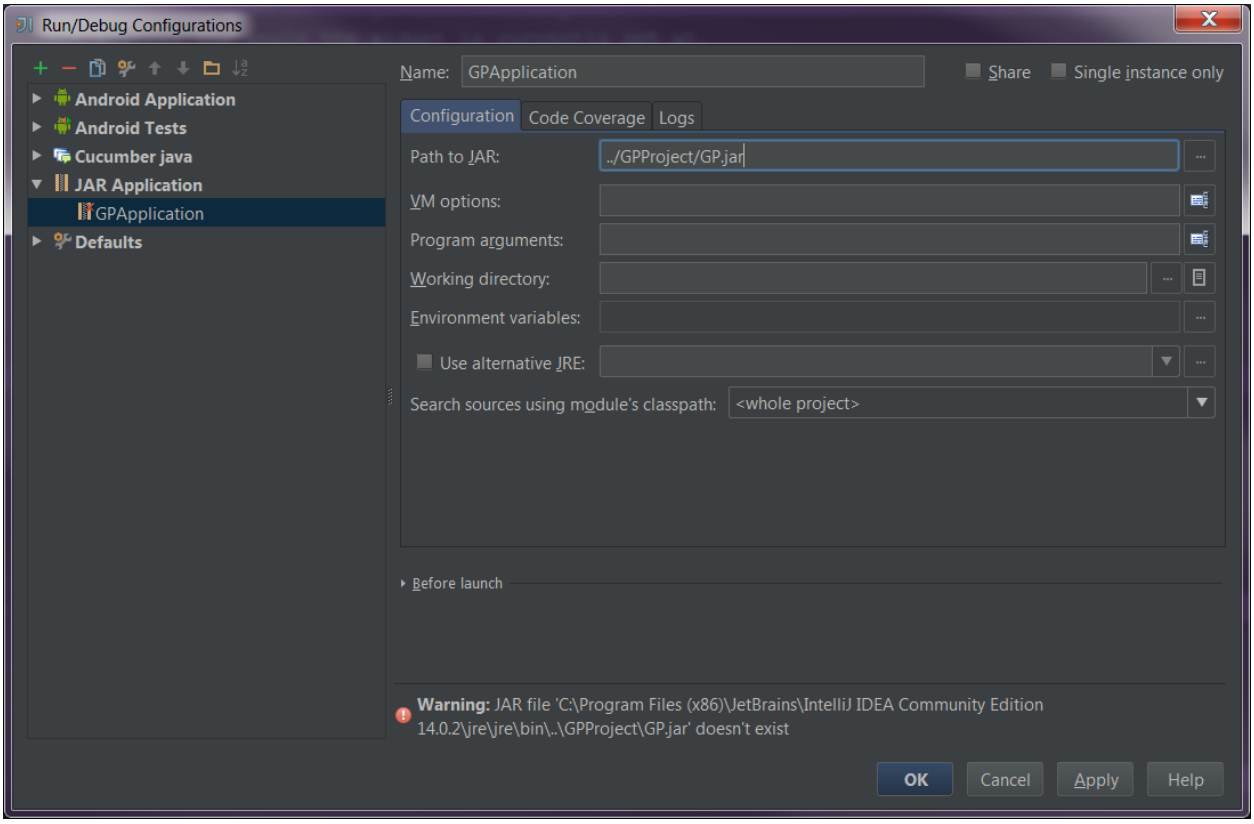
Switch director following the following relative path: ..\GeneticProgramming\
build.bat
run.bat

b.Through GUI:

Double click execution run file.

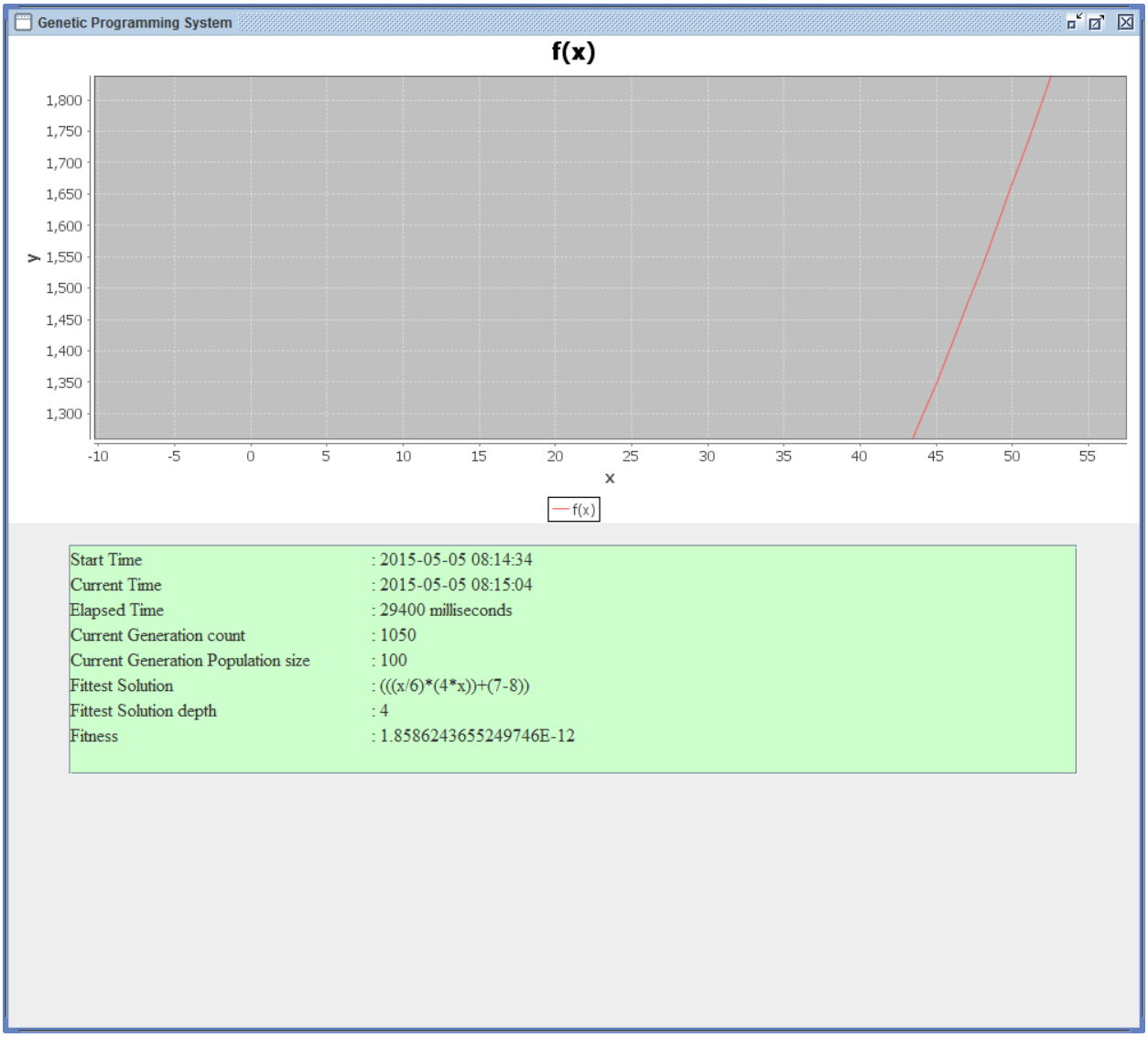
c. Through IntellJ IDE

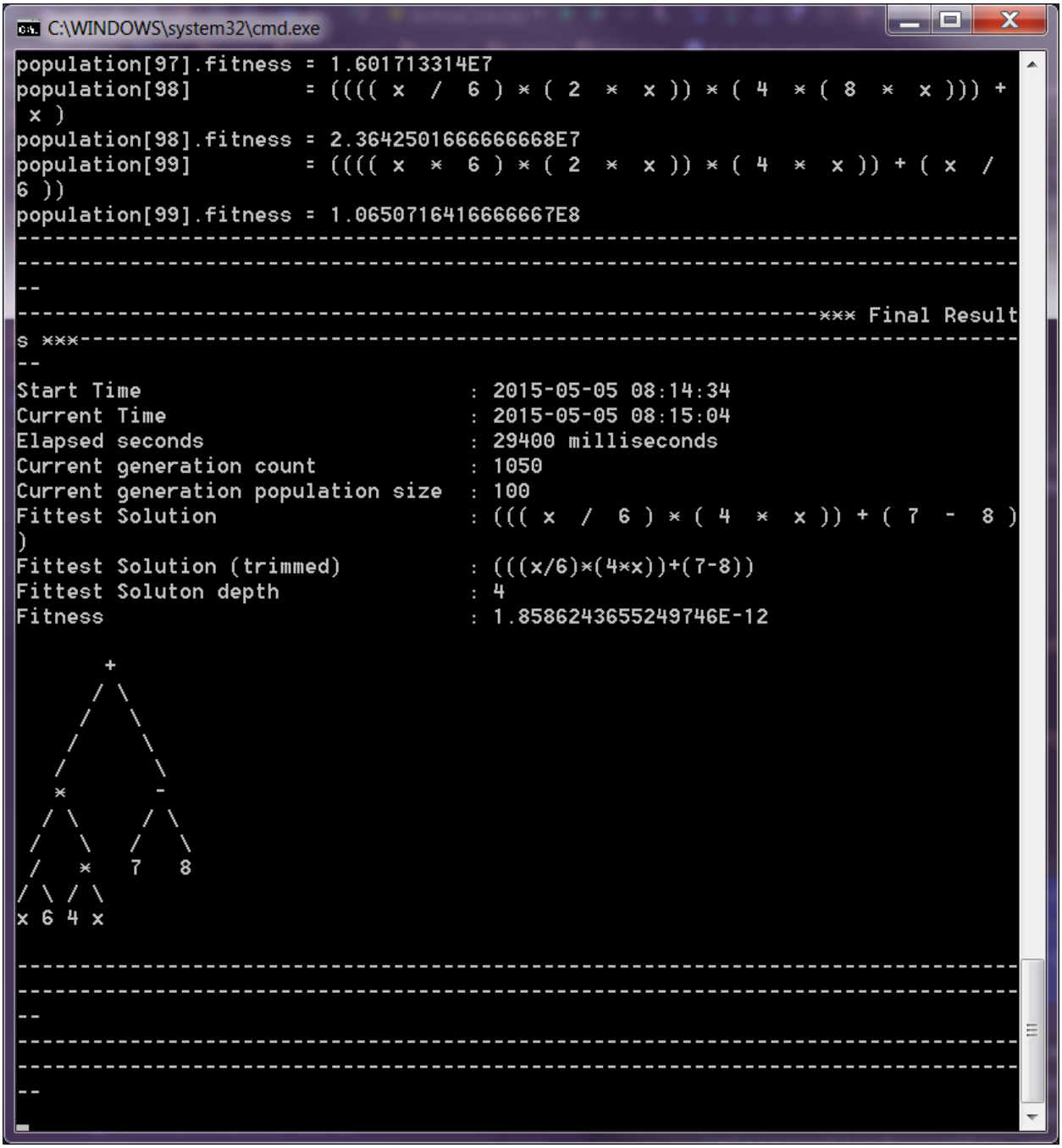
Run-> Edit Configuration ->Jar Application



4. Output

Graphical display and command line


Genetic Programming System
f(x)
1,800
1,750
1,700
1,650
1,600
1,550
1,500
1,450
1,400
1,350
1,300
y
-10
-5
0
5
10
15
20
25
30
35
40
45
50
55
x
f(x)
Start Time : 2015-05-05 08:14:34
Current Time : 2015-05-05 08:15:04
Elapsed Time : 29400 milliseconds
Current Generation count : 1050
Current Generation Population size : 100
Fittest Solution : (((x/6)*(4*x))+(7-8))
Fittest Solution depth : 4
Fitness : 1.8586243655249746E-12


C:\WINDOWS\system32\cmd.exe
population[97].fitness = 1.601713314E7
population[98] = (((((x / 6) * (2 * x)) * (4 * (8 * x))) +
x)
population[98].fitness = 2.3642501666666668E7
population[99] = (((((x * 6) * (2 * x)) * (4 * x)) + (x /
6))
population[99].fitness = 1.0650716416666667E8
*** Final Results ***
Start Time : 2015-05-05 08:14:34
Current Time : 2015-05-05 08:15:04
Elapsed seconds : 29400 milliseconds
Current generation count : 1050
Current generation population size : 100
Fittest Solution : (((x / 6) * (4 * x)) + (7 - 8)
)
Fittest Solution (trimmed) : (((x/6)*(4*x))+(7-8))
Fittest Soluton depth : 4
Fitness : 1.8586243655249746E-12
+
* -
7 8
x 6 4 x

References

1. **Information used requirement analysis**
   [B1] Software Engineering Theory and Practice, Fourth Edition
   Author: Shari Lawrence Pfleeger, Joanne M. Atlee
2. **Information used for the design and development of the genetic operators.**

   [B2] The GP Tutorial. (1996-2013). [Online]. Available:
   http://www.geneticprogramming.com/Tutorial/
3. **Information used for the design and development of the genetic operators.**

   [B3] Kevin Dolan. (2009). "Selection." *Genetic Programming Source*. [Online]. Available:
   http://geneticprogramming.us/Selection.html
4. **Information used for the design and development of the genetic operators.**

   [B4] The Java Tutorial. (19995-2013). Lesson: Algorithms. [Online]. Available:
   http://docs.oracle.com/javase/tutorial/collections/algorithms/#sorting