University of St. Thomas

Graduate Programs in Software

SEIS 610: Software Engineering Fall 2013

Genetic Programming System

Final Report

12/11/2013

Mohammed Alsaihati
Warsame Bashir
Kholoud Benderaa
Rajesh Gupta
Jayesh Naithani

Table of Contents

Project Description

The purpose of this project is to build and design a system that selects a function, which closely matches a predefined “target function”. The “functions”, are derived mathematical equations that use a set of operators and operands to return a final 'y' value. Genetic programming concepts such as 'mutation' and 'crossover' will be used to evolve a learning system as it goes through several generations. The goal is to produce a final equation after evaluating, comparing, and transforming several random functions.

Requirements have been created to determine the closest matching function based on a threshold amongst many other settings. The solution will provide a fitness evaluation method, an initial population generator and several other key methods that lead up to the final product.

Requirements Analysis

- To design and build a Genetic Programming System
- To build the system using current Software Engineering practices
- Find optimal or near optimal solution for target function
    - y = (-3X2 + 7) / 2
    -
- Use genetic programming concepts
- 15 Minute Limitation

Describe Functionality

What will the system do?

- The system will select a solution from a potentially vast solution space which closely matches a predefined target solution.

How will the system do it?

- The Genetic Programming (GP) system will randomly initialize a population of solutions, evaluate the fitness of each solution, and select a solution that meets a pre-determined solution criteria.

- If the solution fitness criteria is not met, then the GP system will generate a new population by selecting solutions from the initial population and by generating a new population as well as performing selection, crossover, and mutation functions to generate a new solution set for performing and repeating the selection process

Where does the system get its data from?

- The system is provided a predefined target solution in the form of training data that a user can enter via a console, or that can be read from a file.

- We will impose a training data set size in order not to impact the execution

performance of the GP system.

Does the system require configuration related settings and what are they?

- The system is also provided several system Genetic Programming System settings.

- These settings define the initial population size, crossover and mutation selection percentages, crossover and mutation function probabilities, data structure size settings such as the height of the initial population tree, acceptable fitness level.

- Define a survivable probability to eliminate solutions that have don't have a good chance of surviving re-generation.

What are some pre-defined GP system settings?

- There will be a pre-defined list of operators and operands.

- There will be a maximum execution time limit - 15 minutes - for the system.

How does the system use the data?

- Training Data used to compute fitness.

- Settings are used to configure the Genetic Programming system.

- The two best solutions will always be chosen from the initial/current population to perform crossover and/or mutation functions.

What format will the data come in?

- Text files

Describe all functions of the system at a high-level?

- Initial Population Generation
    - Single Tree Generation
        - Operators

        - Operands

- Fitness Evaluation

- Determine Termination Condition

- Selection from Initial Population
    - Fitness Comparison
    -
- Crossover

- Mutation

- Selection for next generation
    - Our selection process will evaluate all solutions against a survival threshold and only add to the next generation that ones that meet the survival criteria
    - Selection may also introduce new solutions for the next generation

What are all the inputs/outputs for each function (mutator, tree-generator, fitness-comparator, crossover)?

- Initial Population Generation
    - Inputs:
        - Initial tree height
        - Max population size
        - List of operands (0..9, x)
        - List of operators (+ - / *)
    - Output:
        - Initial population (list of trees)

- Fitness Evaluation

- Inputs:
  - Training Data
  - Initial Tree
- Output:
  - List of trees with fitness values

- Determine Termination Condition for Fitness
  - Input:
    - Single Tree
    - Fitness Criteria Setting
  - Output:
    - True or False

- Crossover
  - The crossover function will take as input selection of next generation population, crossover probability
  - The crossover function will return as output the cross-over modified tree generation solutions.

- Mutation
  - The mutation function will take as an input the crossover modified tree generation, mutation probability, the valid operators and any new trees.
  - The mutation function will return as an output the next tree generation

- Selection
  - Input
    - Two random functions
    - Fitness of each random function
  - Output
    - “Better” function will be returned.

Data Gathering

How will the data be calculated and stored (Training Data) ?

- The training data we are going to use less than 60-100 sets of [x,y] values and the range will be between -30 and +50. For accuracy in our 'y' values, up to 7 decimal places will be used as part of the training data set.

- The training data will be calculated, read and stored as a text file. The training data will have defined 'x' values and computed 'y' values, the size of the data set will vary but will range anywhere from 60 to 100 values.

How long will the data be stored ?

- Data should be stored long enough to generate the target function and doesn't need to be stored any further than that.

What are the settings that need to be stored (Probabilities, population-size, fitness-threshold) ?

- Terminal set, Functional set, and Fitness function (See Requirements - Section E).

User Interface / Front-End Requirements

What type of interface will the system have?

- The system will have simple features and users will only be required of very little interaction. The system will be accompanied with documentations that explains every part of the system's functionality. There will be a Graphical User Interface (GUI) with a single start button that runs the simulation and a final report generated when the system is finished.

How will the data be displayed in the interface ?

- Data will be displayed in a report and the report will contain at minimum:
  - The final calculated function as generated by our GP.
  - The iterations including population size, probabilities used, closest functions, fitness evaluation (%) …
  - The final evaluation time as computed by the GP (start and finish time) and if possible the tree generations as graph outputs.

Will there be any Documentation ?

- A user manual will be provided with details in regards to running the application.
  - Overview of compile and build process.
  - Interactions and outputs of the system.
- There will be JavaDocs about the API's and class definitions.

Audience/Users

Who are the main users of this system ?

- The main user is one who wants to find a closest matching function from a target function.

How will the user engage with the system ?

- The user will "click" a single start button and will receive as an output a final solution with metadata.

Reasons in selecting Object-Oriented approach.

Considering the iterative/incremental methodology, the focus on objects and the ability to adapt projects with changing user requirements made us choose the Object-Oriented approach in analyzing, designing and developing the Genetic Programming System project.

	Structured	**Object-Oriented**
Methodology	SDLC	Iterative/Incremental
Focus	Processes	Objects
Risk	High	Low
Reuse	Low	High
Maturity	Mature and widespread	Emerging
Suitable for	Well-defined projects with stable user requirements	Risky large projects with changing user requirements

Design

System Architecture

The Genetic Programming System is developed using object oriented design principles in the Java. programming language. No external tools or utilities were used to build the system.

Design and Development

We choose to build the system from the ground up because this approach would provide us greatest flexibility and control over the system. Also, we felt we would understand better the Software Engineering concepts we would be learning in class if we developed the system on our own.

We were also aware that our system requirements would change towards the end of our project. Developing using an agile methodology along with having complete development control of our system would help us react quickly and effectively to the change.

The tradeoff here was the learning curve involved in understanding Genetic concepts and implementing reasonably efficient Genetic operator functions on our own. There was also a small possibility that we would not be successful in completing the project if we were to develop the system from scratch as not all the team was proficient in the Java programing language. We needed to effectively partition the project work amongst ourselves so others in the team who were not developing could focus on design, testing, and documentation activities.

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

- Overall team familiarity with the language and object oriented development.
- The language supports development of modular object oriented systems
- The language has mature development tools
- 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.

Data Structures

The data in our system consists of binary trees with nodes that contain operator and operand data, and are part of a population of trees. Additionally, a genetic programming tree has the additional fitness property.

Our choice of implementing the tree and population data structures were between using Strings and Arrays and character parsing operations to implement functions, or to use the power of object oriented programming to represent a binary tree data structure that would allow us to flexibly traverse the tree, evaluate it's value, and perform the operations of crossover and mutation. 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 the Java ArrayList structure as it allow us to easily do the following:

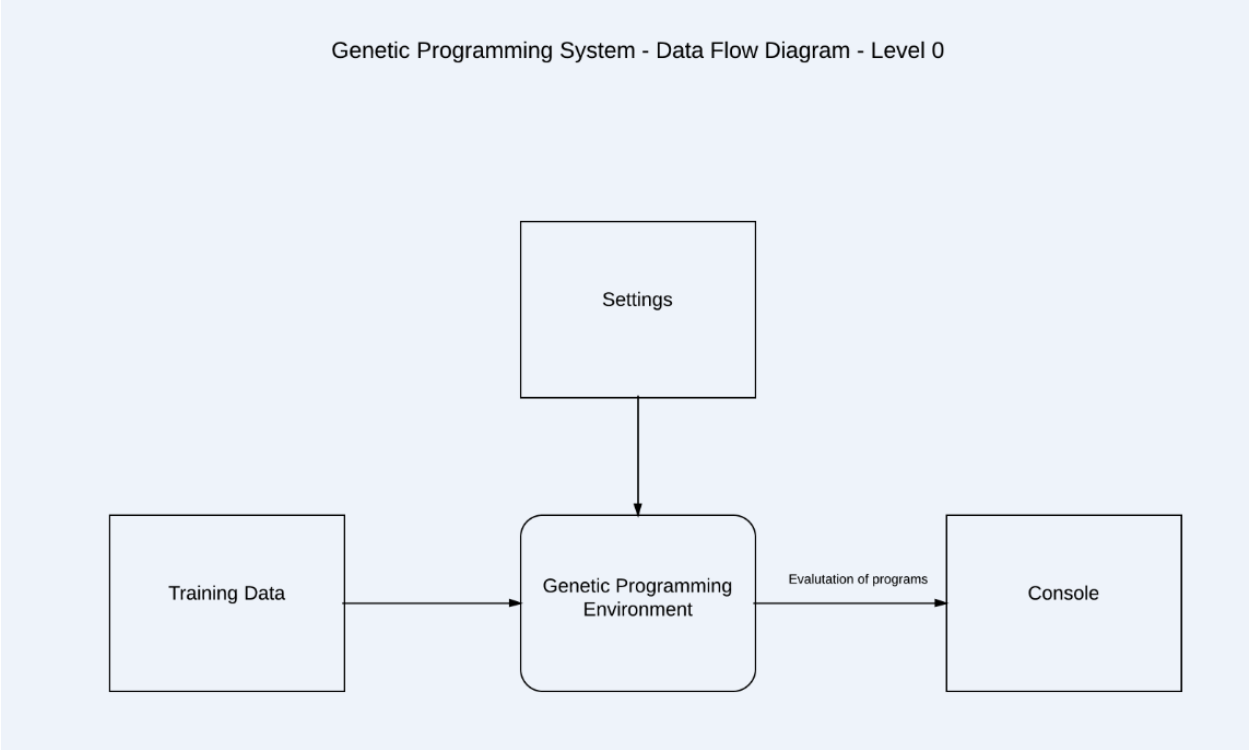
- Iterate over the population and access it using indexes
- Convenient allow us to efficiently sort a population of trees using it's built in sorting capabilities.

Diagrams

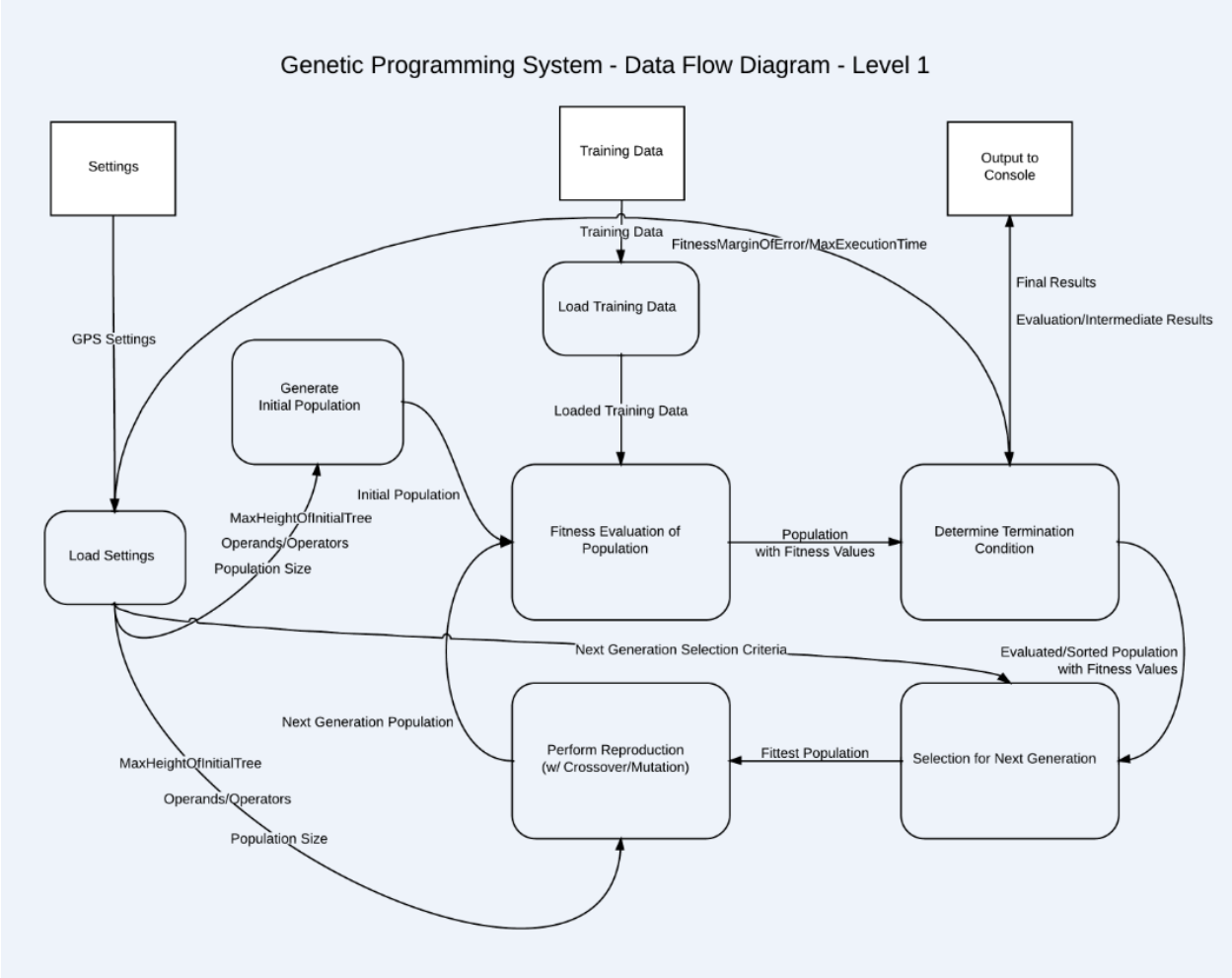
Data-flow diagram

Define all the process diagrams for the main process and sub-processes such as the evolution and crossover processes

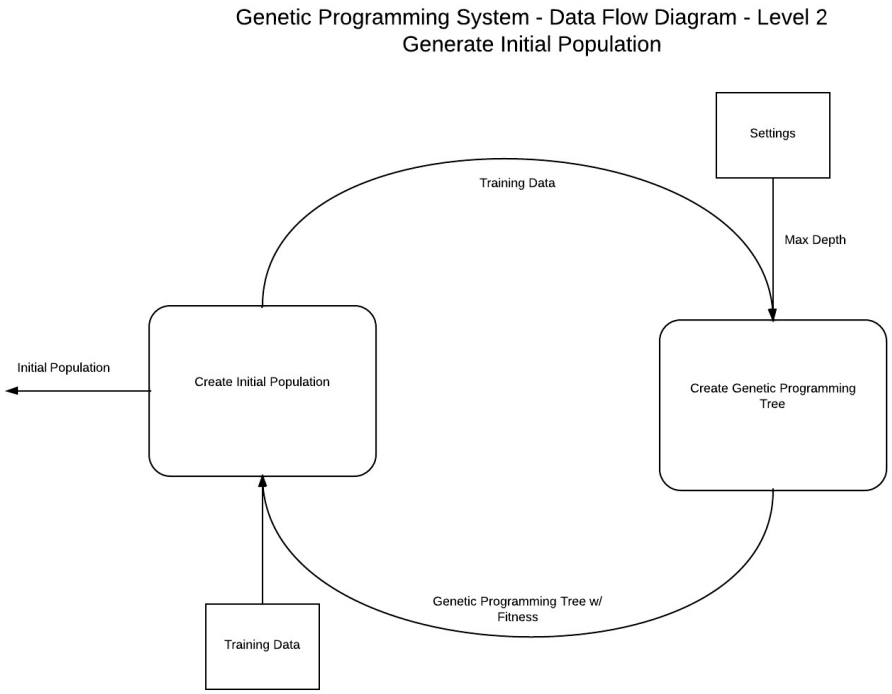
Level 0



Level 1


Genetic Programming System - Data Flow Diagram - Level 1
Settings
Training Data
Output to Console
Training Data
FitnessMarginOfError/MaxExecutionTime
Load Training Data
Final Results
Evaluation/Intermediate Results
GPS Settings
Generate Initial Population
Loaded Training Data
Initial Population
MaxHeightOfInitialTree
Operands/Operators
Population Size
Load Settings
Fitness Evaluation of Population
Population with Fitness Values
Determine Termination Condition
Next Generation Selection Criteria
Evaluated/Sorted Population with Fitness Values
Next Generation Population
MaxHeightOfInitialTree
Operands/Operators
Population Size
Perform Reproduction (w/ Crossover/Mutation)
Fittest Population
Selection for Next Generation

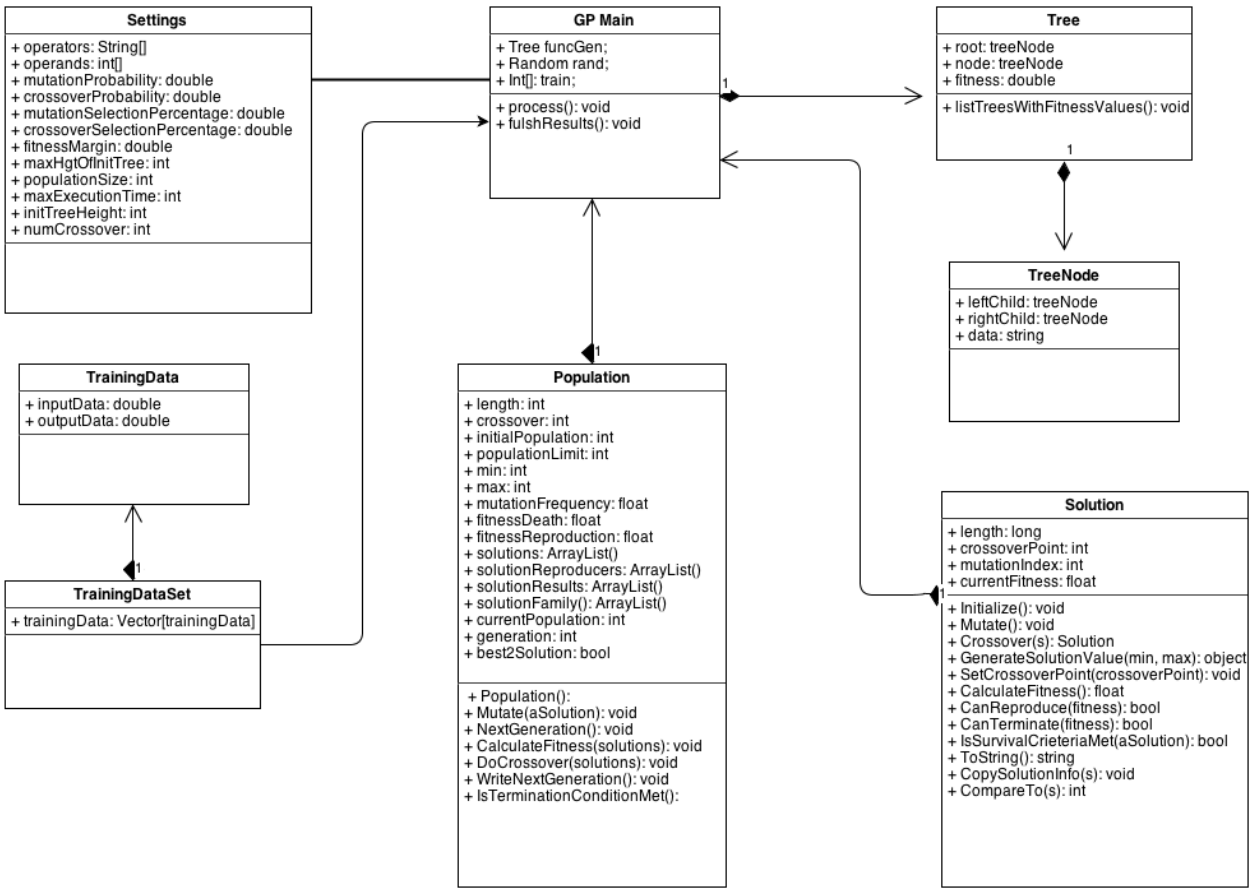
Level 2


Genetic Programming System - Data Flow Diagram - Level 2
Generate Initial Population
Settings
Training Data
Max Depth
Initial Population
Create Initial Population
Create Genetic Programming Tree
Genetic Programming Tree w/ Fitness
Training Data

Class Diagrams (OOD Diagrams)

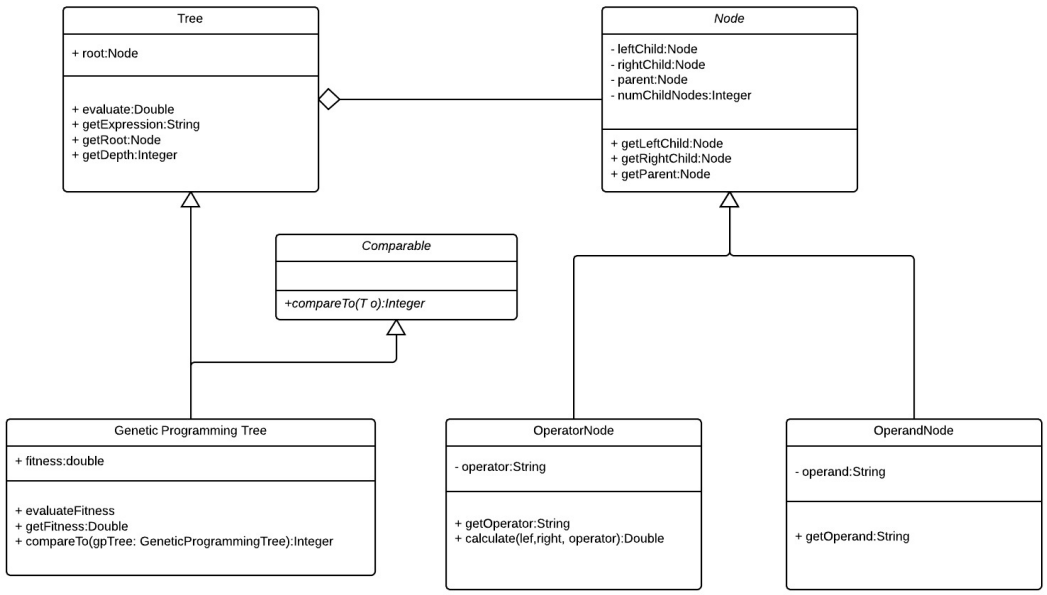
Genetic Programming System

The Genetic Programming (GP) System
Class Diagram

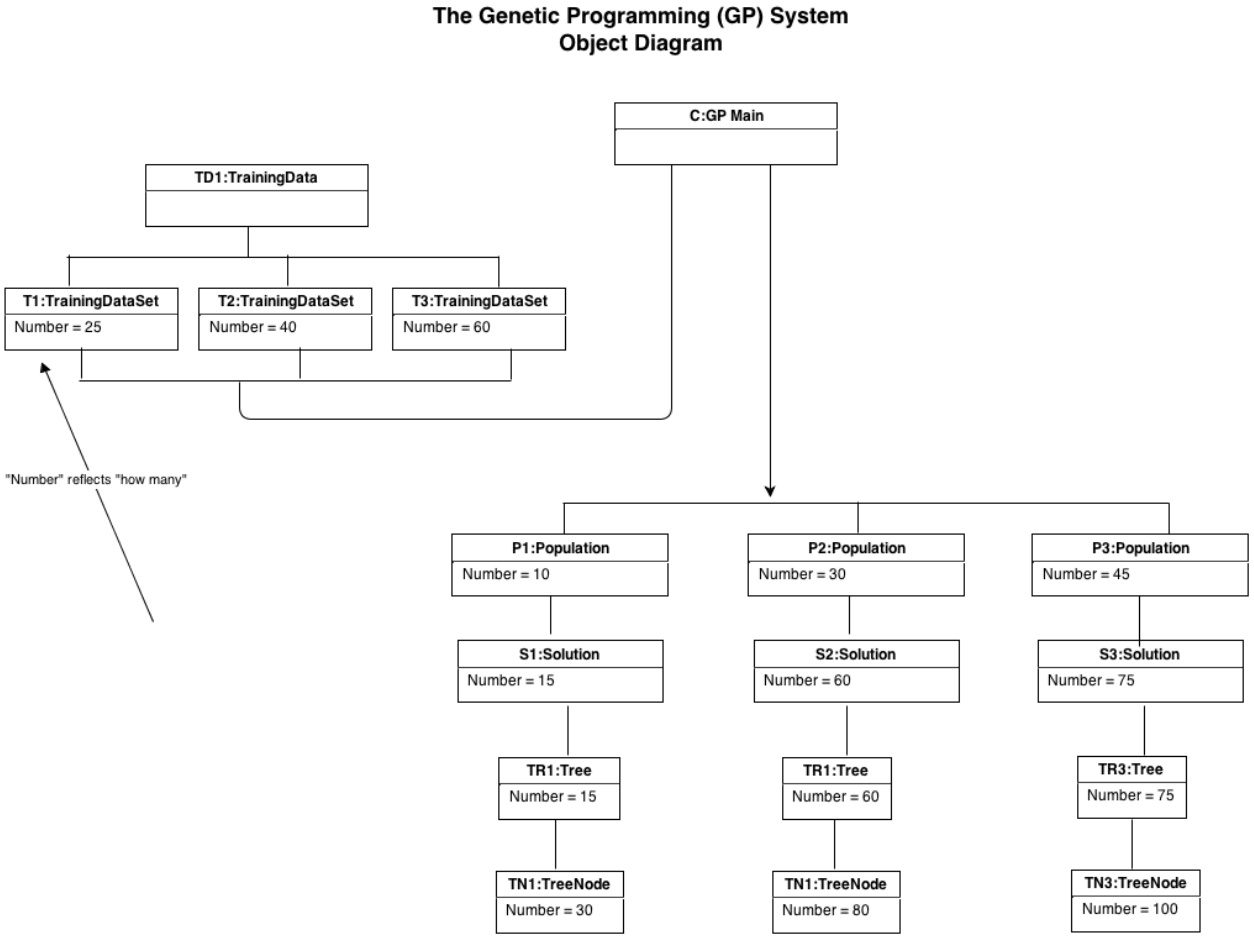


Data Structure Class Diagram

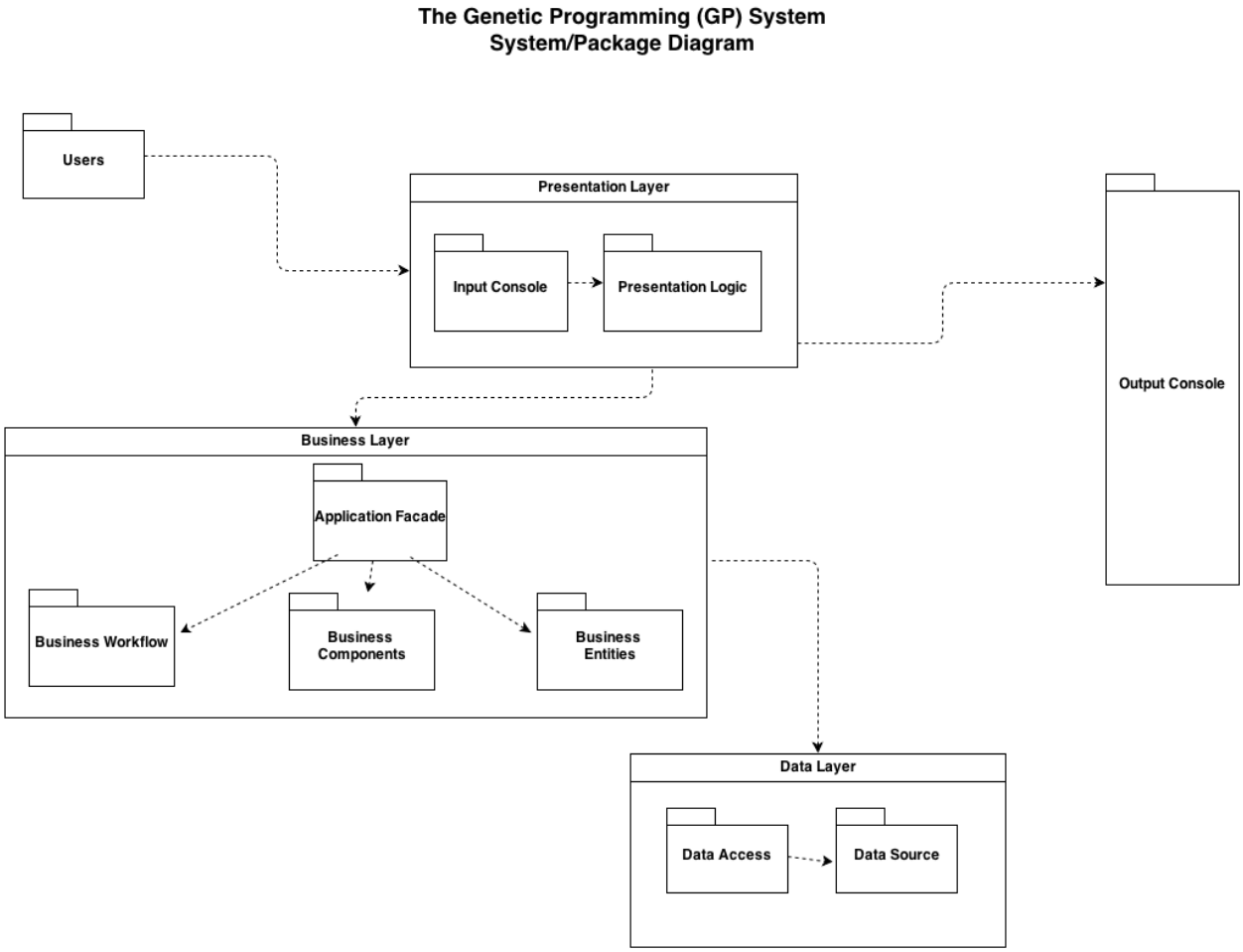
Data Structure Class Diagram



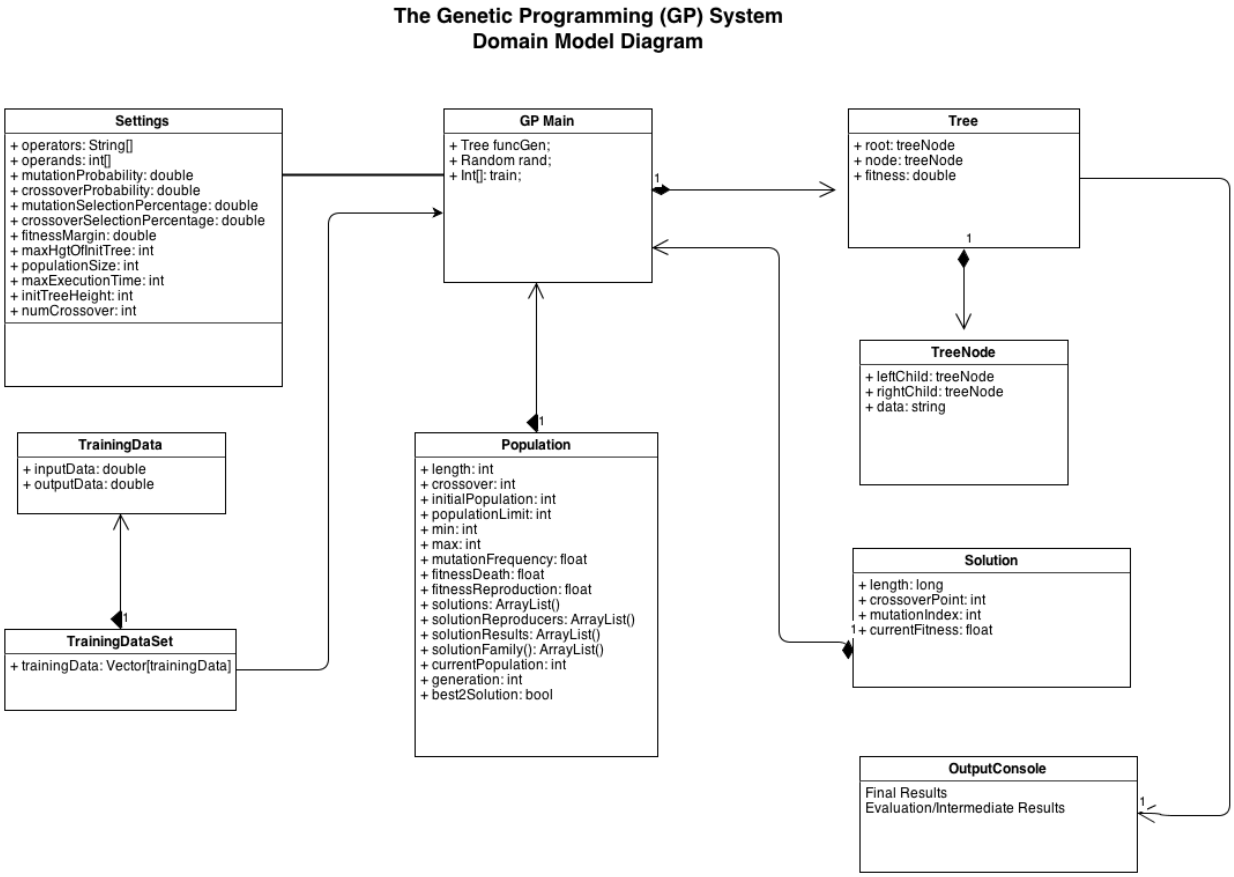
Object Diagram



System/Package Diagram

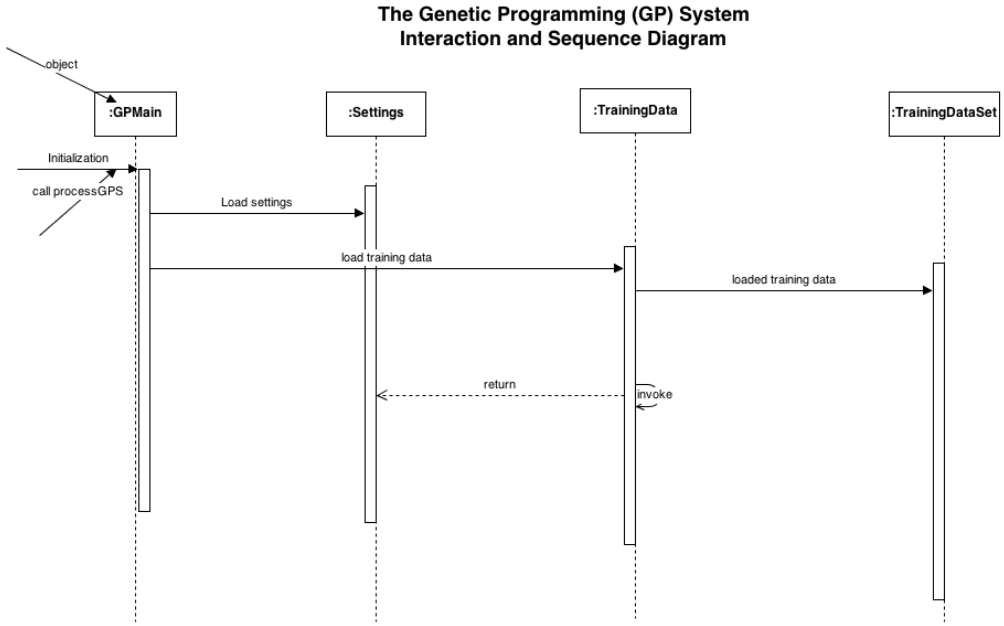


Domain Model Diagram



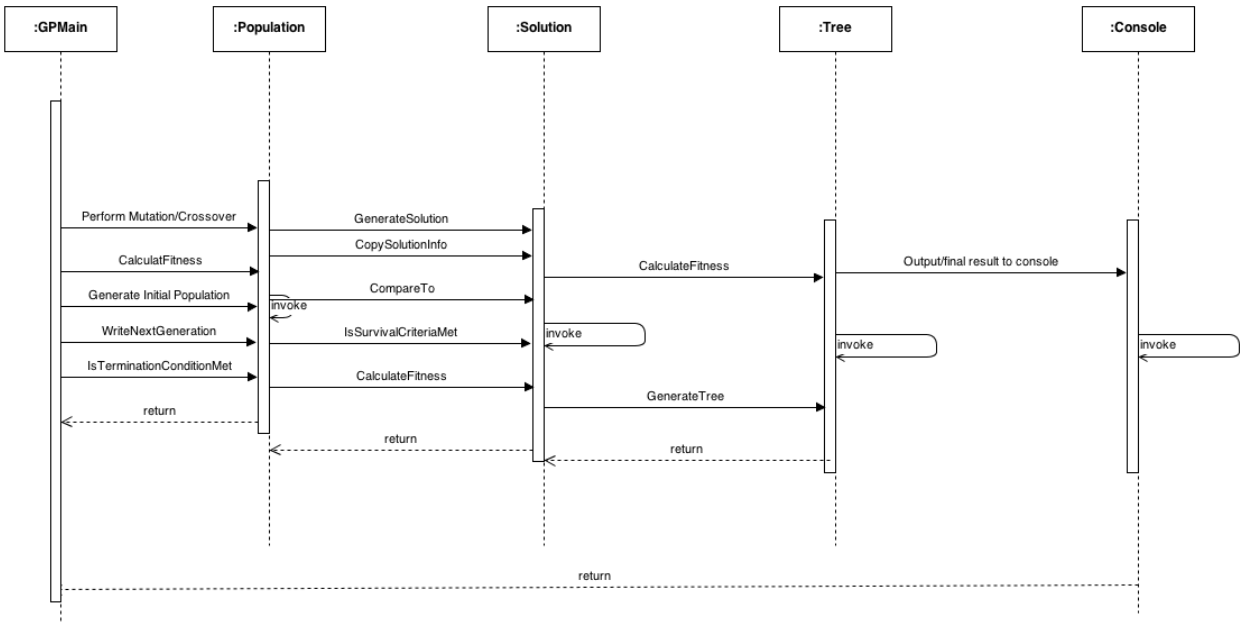
Behaviour Diagrams

Interaction and Sequence Diagram



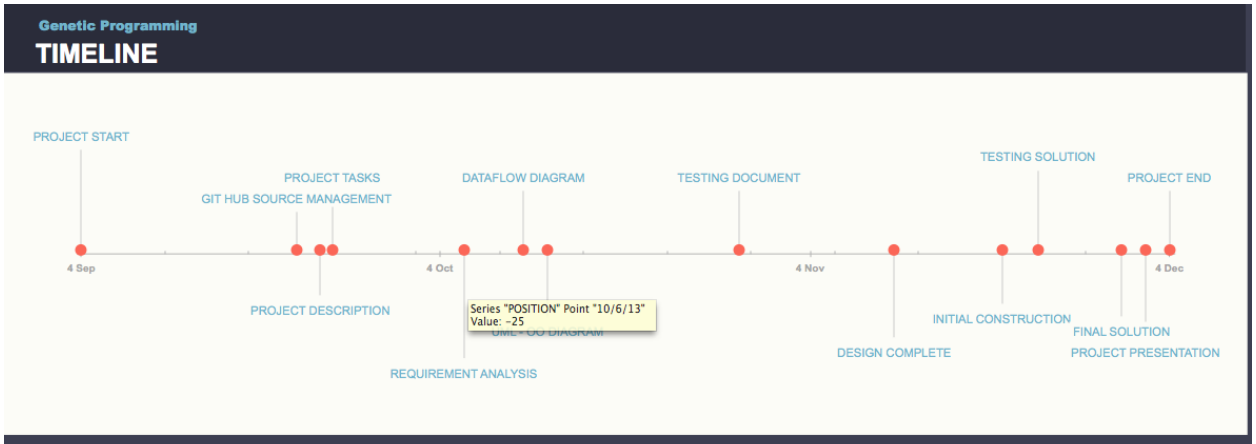
Contd. to page 2 .....

The Genetic Programming (GP) System
Interaction and Sequence Diagram .....(contd. from 1 ....)



Work Breakdown

Project Timeline (Milestones/Estimated Schedule)



PROJECT DETAILS

DATE	MILESTONE	POSITION
9/4/13	Project Start	25
9/22/13	Git Hub Source Management	10
9/24/13	Project Description	-10
9/25/13	Project Tasks	15
10/6/13	Requirement Analysis	-25
10/11/13	Dataflow Diagram	15
10/13/13	UML - OO Diagram	-15
10/29/13	Testing Document	15
11/11/13	Design Complete	-20
11/20/13	Initial Construction	-12
11/23/13	Testing Solution	20
11/30/13	Final Solution	-15
12/2/13	Project Presentation	-20
12/4/13	Project End	15

Task Breakdown

Tasks:

- Project Description **[Warsame Bashir]**
- Requirements and Analysis **[Jayesh, Rajesh]** - **https://blackboard.stthomas.edu/bbcswebdav/pid-1548528-dt-content-rid-4800108_2/courses/201340SEIS610-01/Analysis13F.pdf**
- Preliminary System Design (Due 10/16) **[ALL]**
    - Class Diagrams
    - Interaction Sequence Diagram
    - Behavior/ISE Diagram
    - ERD diagram
    - Flow Diagram
- Project Work Plan (Due 10/16) **[Mohammed, Kholoud]**
- Final System Design (Remaining Items Due: 12/4) **[Warsame, Jayesh, Mohammed]**
- Coding and Development **[Warsame, Rajesh, Jayesh]**
    - Repository setup **[Jayesh]**
    - Development environment/Programming language selection **[Mohammed, Jayesh]**
    - Individual development/coding assignments based on system design **[Kholoud, Rajesh, Warsame]**
    - Build framework **[Warsame, Rajesh]**
    - Unit and system tests **[Kholoud]**
- System Test Plan **[All]**
- Test Results **[Mohammed, Warsame, Rajesh]**
- Lessons Learned **[Jayesh]**
- Project Summary **[Warsame, Kholoud]**
- Generate/Document Weekly SCM Files/Folders **[Rajesh, Mohammed]**
- Visual materials - document printouts, slides **[Rajesh, Kholoud, Warsame]**
- A short user manual **[Jayesh, Mohammed]**
- Post-project analysis **[Mohammed, Rajesh, Jayesh]**
- All project related materials (documents, slides, programs, and executable files

[Warsame, Jayesh, Rajesh]

Total Project Hours

Task Name	Hours	Who
Project Description	1	Warsame
Requirements and Analysis	40	Team
System Design	40	Team
Coding	90	Team
Testing	50	Team
Documentation	30	Team
Presentation	16	Team
Meetings	16	Team

Implementation

- The Java programming language will be used to implement the system.
- The JUnit testing framework will be used to test the system as part of our unit testing plan.
- No external frameworks or plugins other than the JAVA provided namespaces will be used to execute, build and design the functions.
- Refer to User Manual section for program installation and usage information.

Metrics and Measurement

Program Size and Program Coverage Metrics

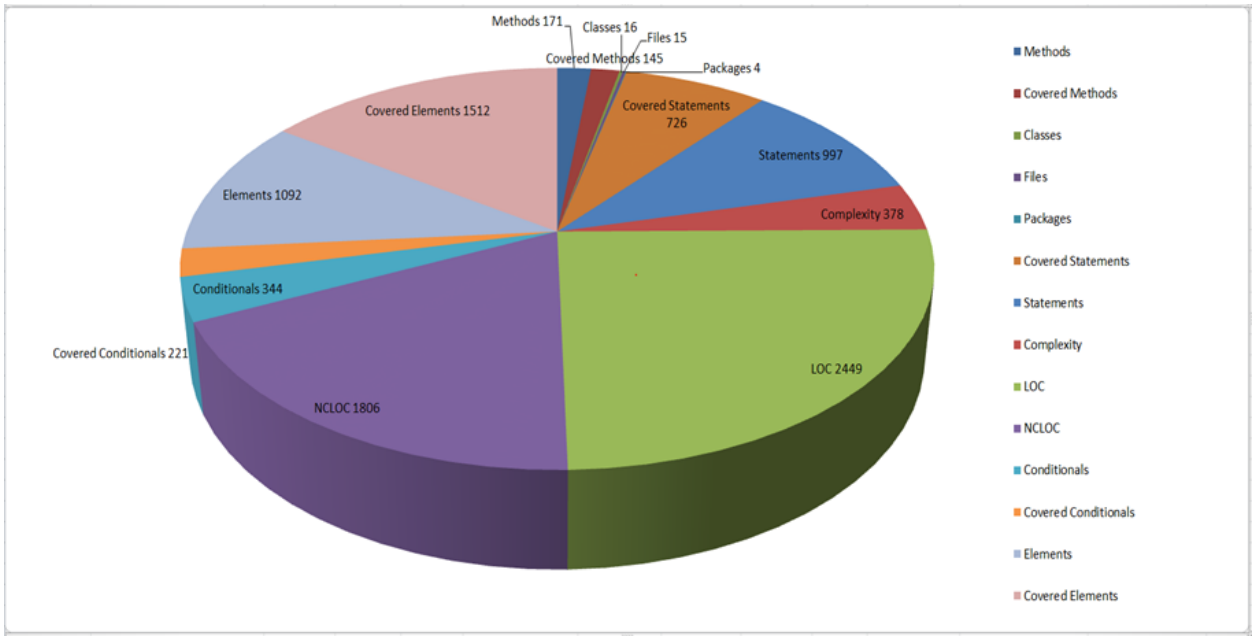
The total number of lines of code (LOC) for the Genetic Programming system were 2449, of which the number of non-commented lines were 1806.

The Genetic Programming system contain 4 packages, 16 classes, and 171 methods. A total number of 145 method were executed during program runs and unit testing.

The total number of statements in the system were 997 of which 726 were covered during program execution and testing. Also, the total number of conditionals were 344 of which 221 were covered.

The total program complexity was 378.

The Eclipse plugin from Atlassian Clover was used to capture these metrics. (https://www.atlassian.com/software/clover)



Complexity

Maximum Cyclomatic Complexity per Class

The maximum complexity value for a method in the data package was 11. This was for swap() method in the Node class which attempts to swap the current node with a given node.

The maximum complexity value of a method in the utilities package was 10, for the crossoverTrees() method.

The default package consisted on the main program with a maximum method complexity of 6. Complexity value of methods below 10 indicates that the Genetic Programming system has mainly well structured and stable methods.

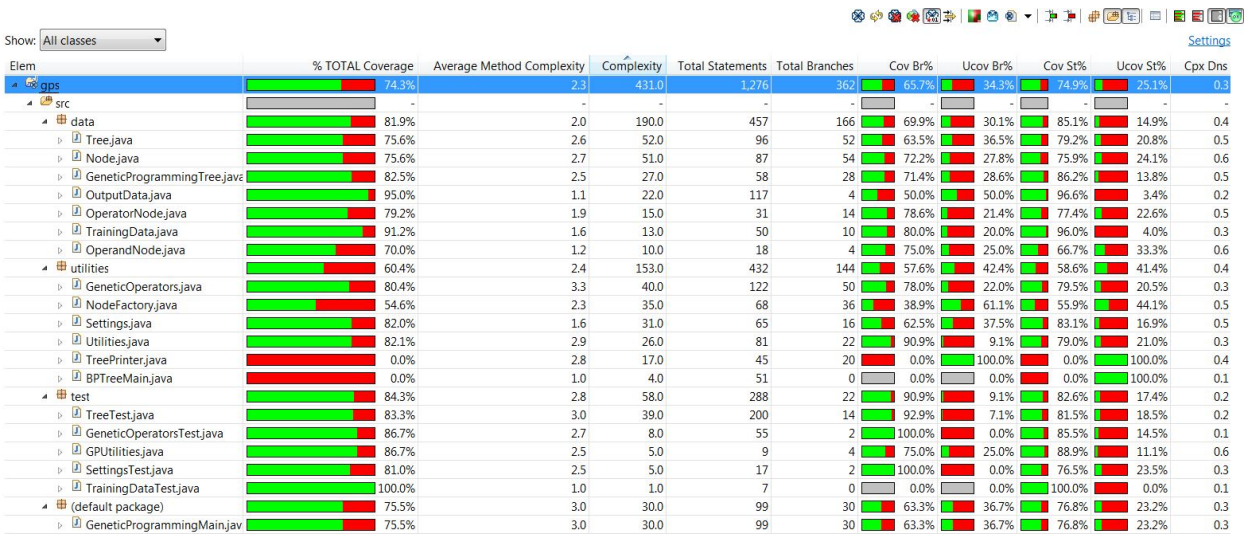
The Eclipse Metrics plugin was used to capture this information. (http://metrics.sourceforge.net/)

Metric	Total	Mean	Std. Dev.	Maximu...	Resource causing Maximum	Method
Afferent Coupling (avg/max per packageFragment)		4.5	4.717	11	/gps/src/utilities	
Number of Interfaces (avg/max per packageFragment)	0	0	0	0	/gps/src	
McCabe Cyclomatic Complexity (avg/max per method)		2.185	1.905	11	/gps/src/data/Node.java	swap
src		2.185	1.905	11	/gps/src/data/Node.java	swap
data		1.929	1.882	11	/gps/src/data/Node.java	swap
Node.java		2.333	2.211	11	/gps/src/data/Node.java	swap
Tree.java		2.5	2.54	11	/gps/src/data/Tree.java	swap
GeneticProgrammingTree.java		2.273	2.093	8	/gps/src/data/GeneticProgrammingTree.ja...	getGeneticTreePopulation
OperatorNode.java		1.875	1.615	6	/gps/src/data/OperatorNode.java	calculate
TrainingData.java		1.625	0.484	2	/gps/src/data/TrainingData.java	generateInitialTrainingData
OperandNode.java		1.25	0.433	2	/gps/src/data/OperandNode.java	OperandNode
OutputData.java		1.1	0.3	2	/gps/src/data/OutputData.java	displayFinalResults
utilities		2.292	2.036	10	/gps/src/utilities/GeneticOperators.java	crossoverTrees
GeneticOperators.java		3.333	2.427	10	/gps/src/utilities/GeneticOperators.java	crossoverTrees
Utilities.java		2.667	2.261	8	/gps/src/utilities/Utilities.java	printTreeNodeInternal
NodeFactory.java		2.333	2.087	8	/gps/src/utilities/NodeFactory.java	getWeightedRandomOperand
TreePrinter.java		2.5	2.5	8	/gps/src/utilities/TreePrinter.java	printNodeInternal
Settings.java		1.632	1.037	4	/gps/src/utilities/Settings.java	debug
BPTreeMain.java		1	0	1	/gps/src/utilities/BPTreeMain.java	test1
(default package)		3	1.897	6	/gps/src/GeneticProgrammingMain.java	process
GeneticProgrammingMain.java		3	1.897	6	/gps/src/GeneticProgrammingMain.java	process
test		2.667	1.247	5	/gps/src/test/GeneticOperatorsTest.java	testCrossover
GeneticOperatorsTest.java		2.667	1.7	5	/gps/src/test/GeneticOperatorsTest.java	testCrossover
TreeTest.java		2.846	1.099	5	/gps/src/test/TreeTest.java	testGenerateInitialTree
GPUtilities.java		2.5	1.5	4	/gps/src/test/GPUtilities.java	testGetRandomNumber
SettingsTest.java		2.5	0.5	3	/gps/src/test/SettingsTest.java	testPrintAllSettings
TrainingDataTest.java		1	0	1	/gps/src/test/TrainingDataTest.java	testLoadTrainingData

Average and Total Cyclomatic Complexity per Class

The following chart provides a view of the total coverage of a methods in all classes, their average method complexity, total number of statements and total branches. The average method complexity of methods does not exceed 3.3, indicating the presence of structured and stable code in the methods.

(https://www.atlassian.com/software/clover)



Show: All classes — Settings

Elem	% TOTAL Coverage	Average Method Complexity	Complexity	Total Statements	Total Branches	Cov Br%	Ucov Br%	Cov St%	Ucov St%	Cpx Dns
gps	74.3%	2.3	431.0	1,276	362	65.7%	34.3%	74.9%	25.1%	0.3
src	-	-	-	-	-	-	-	-	-	-
data	81.9%	2.0	190.0	457	166	69.9%	30.1%	85.1%	14.9%	0.4
Tree.java	75.6%	2.6	52.0	96	52	63.5%	36.5%	79.2%	20.8%	0.5
Node.java	75.6%	2.7	51.0	87	54	72.2%	27.8%	75.9%	24.1%	0.6
GeneticProgrammingTree.java	82.5%	2.5	27.0	58	28	71.4%	28.6%	86.2%	13.8%	0.5
OutputData.java	95.0%	1.1	22.0	117	4	50.0%	50.0%	96.6%	3.4%	0.2
OperatorNode.java	79.2%	1.9	15.0	31	14	78.6%	21.4%	77.4%	22.6%	0.5
TrainingData.java	91.2%	1.6	13.0	50	10	80.0%	20.0%	96.0%	4.0%	0.3
OperandNode.java	70.0%	1.2	10.0	18	4	75.0%	25.0%	66.7%	33.3%	0.6
utilities	60.4%	2.4	153.0	432	144	57.6%	42.4%	58.6%	41.4%	0.4
GeneticOperators.java	80.4%	3.3	40.0	122	50	78.0%	22.0%	79.5%	20.5%	0.3
NodeFactory.java	54.6%	2.3	35.0	68	36	38.9%	61.1%	55.9%	44.1%	0.5
Settings.java	82.0%	1.6	31.0	65	16	62.5%	37.5%	83.1%	16.9%	0.5
Utilities.java	82.1%	2.9	26.0	81	22	90.9%	9.1%	79.0%	21.0%	0.3
TreePrinter.java	0.0%	2.8	17.0	45	20	0.0%	100.0%	0.0%	100.0%	0.4
BPTreeMain.java	0.0%	1.0	4.0	51	0	0.0%	0.0%	0.0%	100.0%	0.1
test	84.3%	2.8	58.0	288	22	90.9%	9.1%	82.6%	17.4%	0.2
TreeTest.java	83.3%	3.0	39.0	200	14	92.9%	7.1%	81.5%	18.5%	0.2
GeneticOperatorsTest.java	86.7%	2.7	8.0	55	2	100.0%	0.0%	85.5%	14.5%	0.1
GPUtilities.java	86.7%	2.5	5.0	9	4	75.0%	25.0%	88.9%	11.1%	0.6
SettingsTest.java	81.0%	2.5	5.0	17	2	100.0%	0.0%	76.5%	23.5%	0.3
TrainingDataTest.java	100.0%	1.0	1.0	7	0	0.0%	0.0%	100.0%	0.0%	0.1
(default package)	75.5%	3.0	30.0	99	30	63.3%	36.7%	76.8%	23.2%	0.3
GeneticProgrammingMain.jav	75.5%	3.0	30.0	99	30	63.3%	36.7%	76.8%	23.2%	0.3

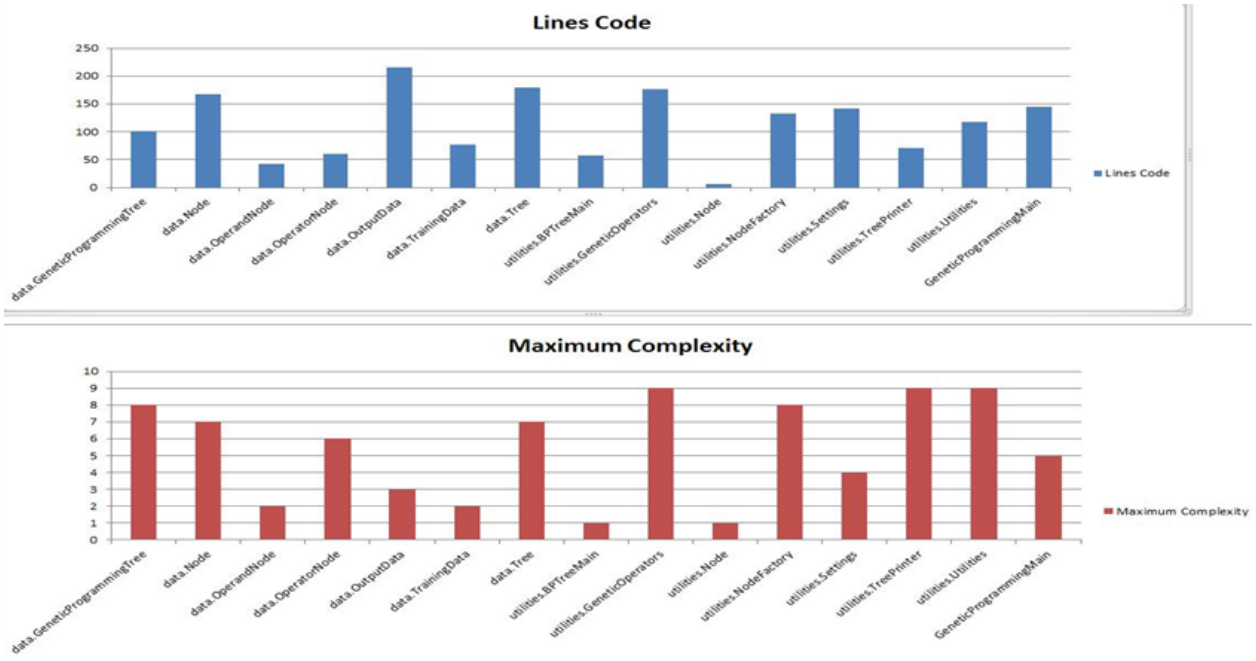
Lines of Code vs. Maximum Complexity

The following shows the LOC versus the maximum complexity at the Class (and not Method) level.

The most lines of code (>200) were in the class used to gather and display the output data. However, the total complexity of this class was a low value of 3. None of the other classes exceeded more than 200 lines of code, well below the max value of 750 when a class should be split further to delegate responsibilities.

The greatest complexity was found in the Genetic Programming Tree, Tree, and GeneticOperators classes. All complexity numbers were less than 9 indicating structured and stable code once again.

These numbers were collected using the static analysis Understand tool from http://www.scitools.com/.

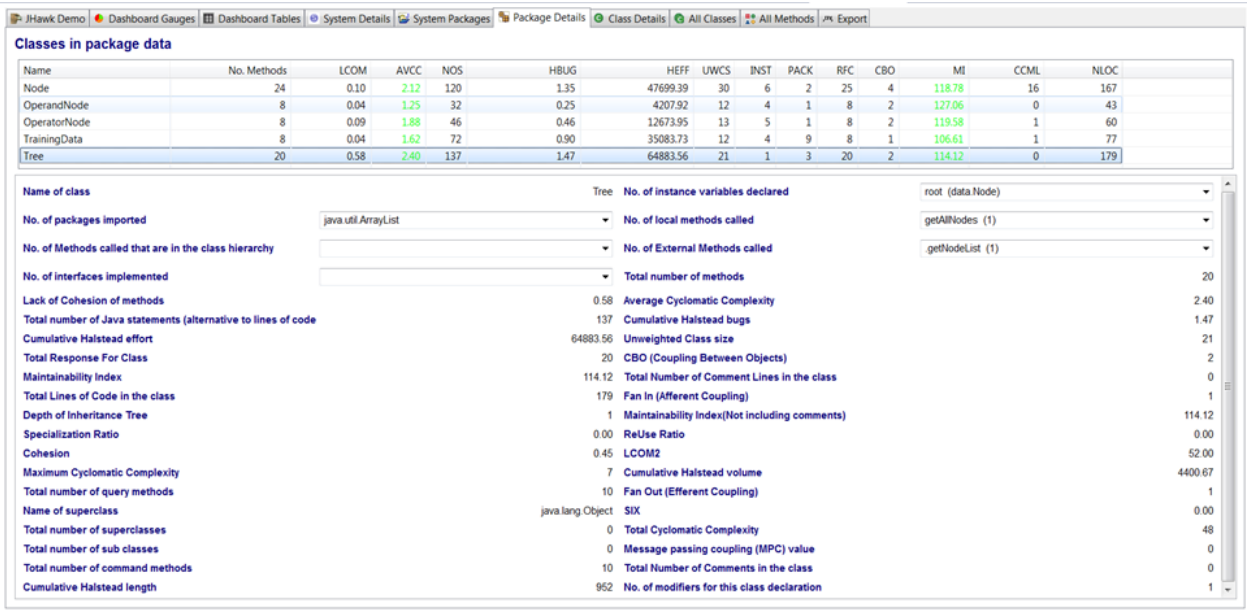


Size Complexity and Maintainability Index for classes in the Data package

The limited functionality version of the Eclipse plugin provided by http://www.virtualmachinery.com/jhawkprod.htm was used to capture the following size and maintainability index metrices for classes in the Gentic Programming system data package.

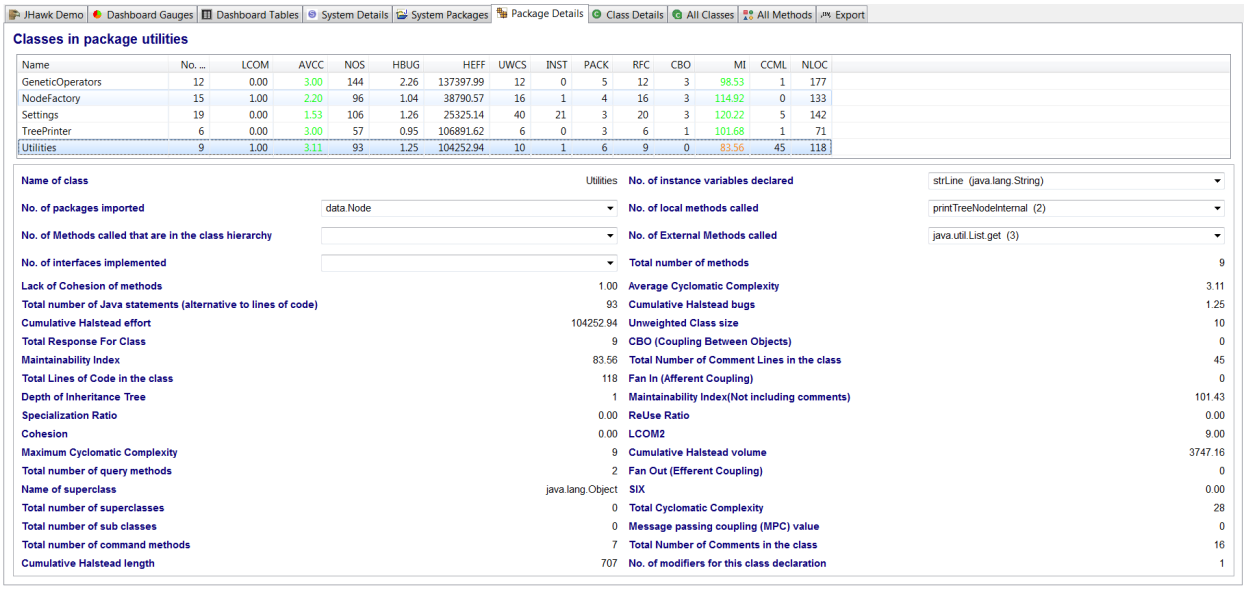
src/data

The Maintainability Index scores of the top 5 classes in the data package ranged between 106 and 127, all well above the 85 threshold value indicating the existence of highly maintainable code.



src/utilities

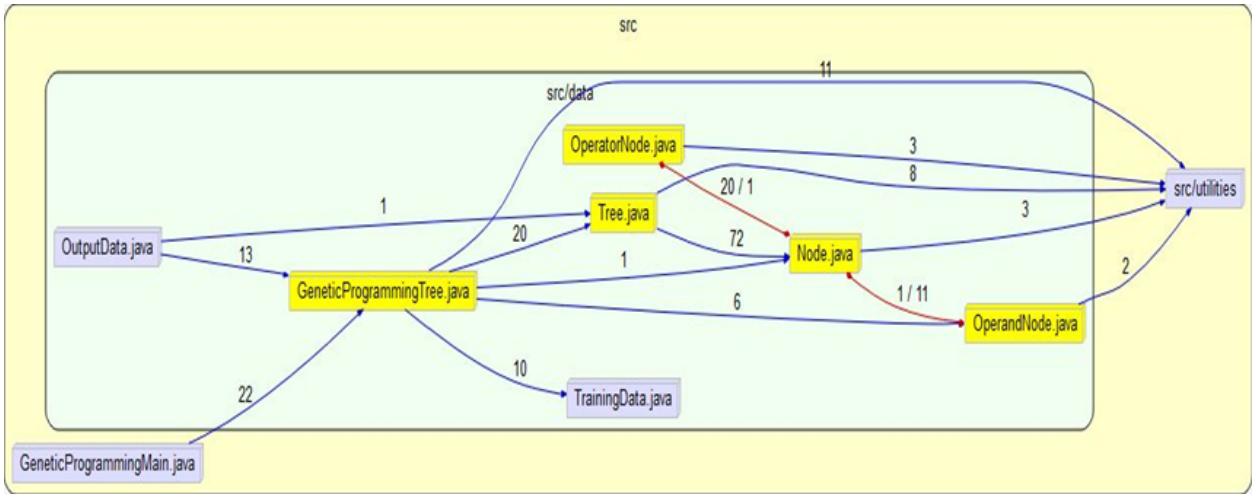
The Maintainability Index scores of the top 5 classes in the utilities package ranged between 83.56 and 120.22.. The low score was for the Utilities class, which was contained method for printing the tree on the command line console. Surprisingly, the Maintainability Index score for this class not including comments was 101.43. Generally comments help in increasing the Maintainability Index. However not so in this case



Dependency Call Graph

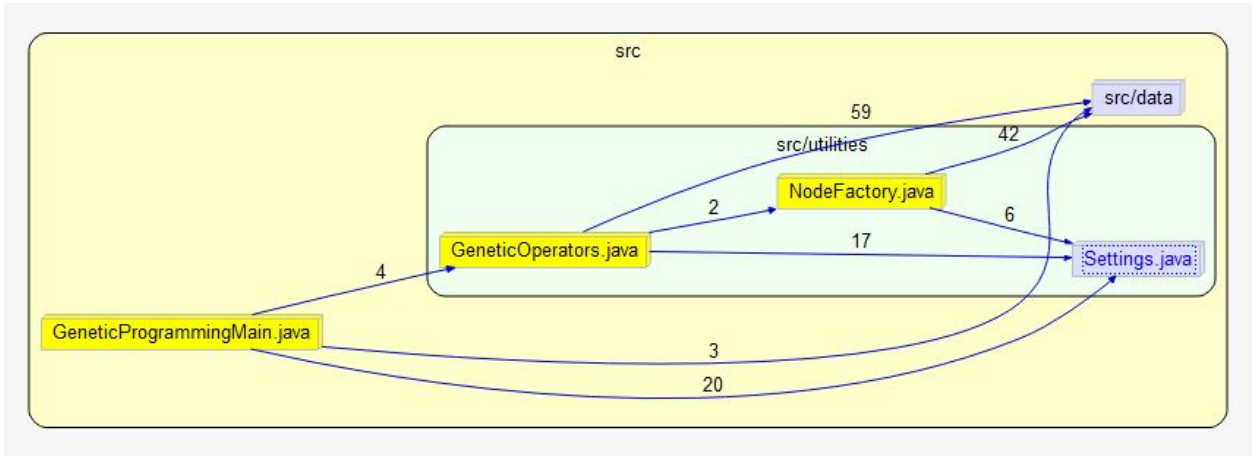
src/data

The following graph shows call dependencies between classes in the data package. The numbers on the edges indicate the number of incoming calls into a class. The Node class has the most incoming calls from the Tree class below.



src/utilities

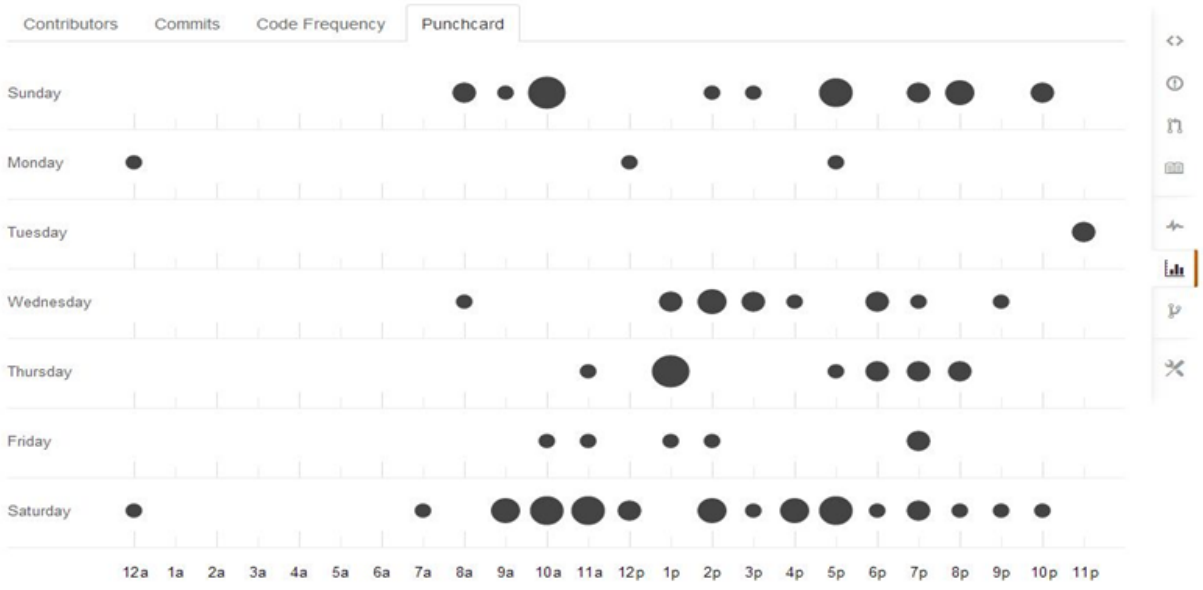
The NodeFactory calls that is responsible to randomly create a terminal or function node has the maximum number of calls to data classes.



SCM Activity on GitHub

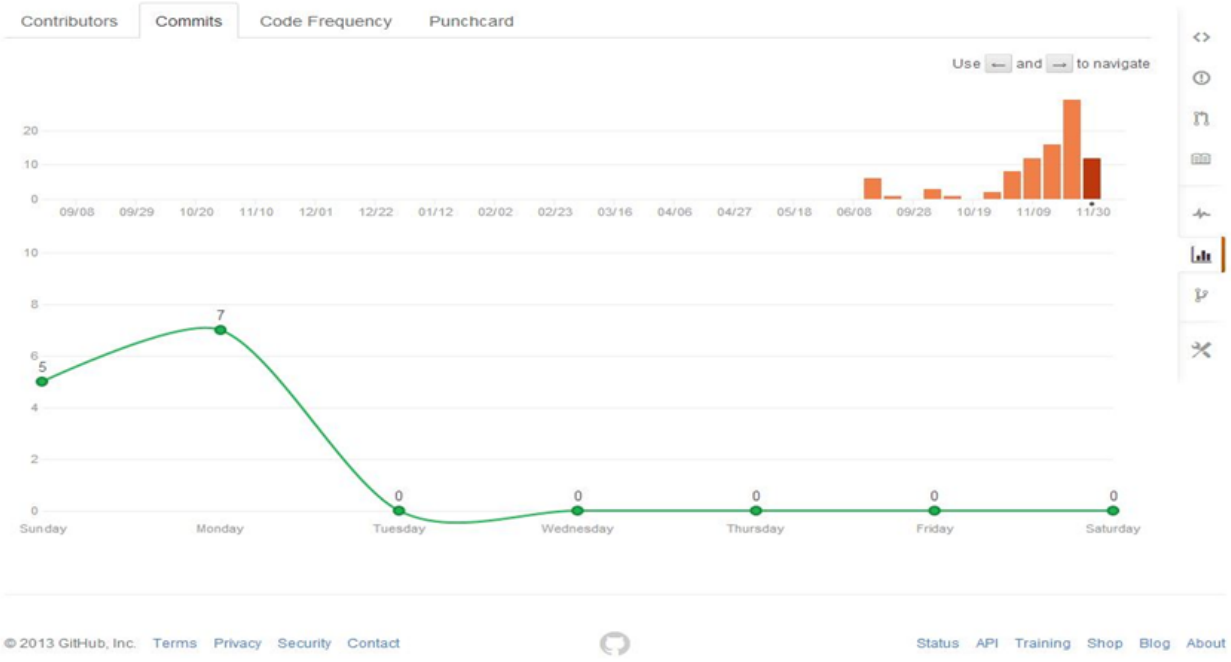
Repository activity by weekday

The punch card graph below generated from GitHub indicates the activity in the Genetic Programming SCM repository during week days. Heaviest activity was usually during weekends. Least amount of activity was recorded on Mondays and Tuesdays.



Monthly commit frequency

The graph below indicates the repository activity from a monthly perspective. The graph indicates a heavier activity in the month of November, few weeks before the end of the project.



Test Plan and Results

Description

This test plan details each test case using the white-box and black-box testing methodologies. An explanation of how testing is conducted and relationships between program functionality is given. This document also contains sample test run metrics and the results received.

How Testing was conducted.

White-Box structural testing was used to verify the majority of the program. Selective paths were chosen due to resource constraints and lack of automated test plan. Most testing was done as unit-testing and the core genetic programming functionality served as the base for the other internal tests. Following the premise to choose quality test paths and not conduct *exhaustive testing,* key inputs were chosen. Black-box testing was also used to verify the genetic programming output with graphing and other visualization tools.

Test Plans

Name: **'Crossover_Test'**

Description: This test represents the crossover functionality. Crossover is a means of combining genetic material of two parents (trees) by swapping part of one parent (tree) with another parent (tree).

Input: 2 single trees with a branch of 'n' nodes in each tree.
Tree 1: [3 2 + 3 3 +], Tree 2: [4 3 + 3 2 /]

Expected Output: A swap of nodes between one tree and another.

Known Issues: None

Test Status: Pass / Fail

Name: **Fitness_Tree_Test**

Description: This test makes sure a fitness is evaluated for a single tree. If a tree is invalid due to an evaluate expression issue such as the ‘divide by zero’ then the test fails. If a fitness is returned that is too large to be represented by a java double then this test also fails.

Input: A single tree with 3 nodes (an ‘x’ operand node is present)
[1 x +]
Expected Output: The expected output is a

Known Issues: None

Test Status: Pass / Fail

Name: **Sort_Selection_Test**

Description: This test will sort a list of Tree objects by fitness and verifies the sort against predefined items matching in ascending order.

Input: A randomly generated population of tree objects with calculated fitnesses.

Expected Output: The expected output is a sorted population of trees in ascending order

Known Issues: None

Test Status: Pass / Fail

Name: **Sort_Selection_Test**

Description: This test will sort a list of Tree objects by fitness and verifies the sort against predefined items matching in ascending order.

Input: A randomly generated population of tree objects with calculated fitnesses.

Expected Output: The expected output is a sorted population of trees in ascending order

Known Issues: None

Test Status: Pass / Fail

Name: **Mutation_Test**

Description: This test checks to see if a random node in a predefined tree has changed it's before and after value.

Input: A single tree object with a predefined value of either an operator, operand or 'x' operand.

Expected Output: The same tree object with a random item of the same type (operator, operand [x])

Known Issues: None

Test Status: Pass / Fail

Name: **Selection_Test**

Description: This test uses a fixture of probabilities such as the selection probability to make sure the number of trees returned meet the predefined criteria.

Input: A randomly generated population of trees sorted by fitness

Expected Output:

Known Issues: None

Test Status: Pass / Fail

Name: **Divide_By_Zero_Test**

Description: This test is verifying the tree 'evaluate' function for a specific scenario. The test will fail in any 'divide by zero' errors and checks to make sure a valid operand is returned.

Input: A single tree with a left operand child and right operand child of either 'x' or the number zero. The root node is a 'divide operator.

Expected Output: The item/value of the left child.

Known Issues: None

Test Status: Pass / Fail

Name: **Tree_Depth_Test**

Description: This test makes sure tree depth calculation is returned correctly. A tree depth is defined as 1 + number of longest nodes in the tree.

Input: A single tree with 3 nodes. e.g [1 3 +]

Expected Output: The expected output is 2, root node plus the longest path to a child node [1].

Known Issues: None

Test Status: Pass / Fail

Name: **Tree_Max_Height_Test**

Description: This test validates the property: ‘max height’. The test fails when max height has been passed when a tree in a population is generated.

Input: A single empty tree object

Expected Output: Height of tree is less than ‘max height’ definition.

Known Issues: None

Test Status: Pass / Fail

Name: **Generate_Tree_Test**

Description: This test covers the tree generation functionality and uses the size of the population

to validate that trees have been created.

Input: An empty ArrayList.

Expected Output: A non-null array list with a size greater than 1.

Known Issues: None

Test Status: Pass / Fail

Name: **Test_Tree_Copy**

Description: This test will validate a copy function making sure an empty object is no longer null and has valid attributes after executing the copy function.

Input: An empty tree object with no child nodes.

Expected Output: A populated tree object with ‘n’ children and a calculated fitness.

Known Issues: None

Test Status: Pass / Fail

Name: **Node_Tests**

Description: This test will verify the node level internal operations such as ‘Getting Operator Nodes’ and ‘ Getting Operand Nodes’.

Input: A single tree with a 2 operand nodes and 1 operator node. [1 2 +]

Expected Output: Get-Operators should return 1. Get-Operands should return 2

Known Issues: None

Test Status: Pass / Fail

Name: **Program_Time_Complete [Black-box]**

Description: This test will verify the program finishes in the allotted time of 15 minutes.

Input: A settings change in the ‘settings.properties’ file to modify the ‘max-depth’

Expected Output: Program time exceeds the ‘max-time’ limit.

Known Issues: None

Test Status: Pass / Fail

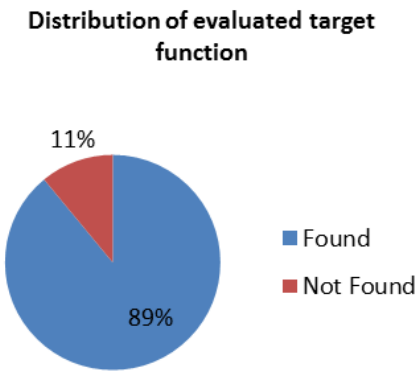
Test Metrics

The genetic programming system was stressed tested to see when run 100 ‘times, what the distribution of **‘found’** and ‘**not found’** would be like. The test was to determine how the program was able to find a solution that matches the target function.

Total # of requests: 100

Target Function: y = (-3X2 + 7) / 2

RESULTS:



Explanation of Results

The test reduced the numeral property in the program settings file **'maxgenerationcount'** to 100**.** This forced the application to not eagerly find a target function sooner and to produce a more realistic 'run'. In the sampling used, the program was able to find the target function in 60-100 generations.

Testing Requirements

The core required test cases for the unit tests were created and **linked** using GitHub (see table 1) to track requirements against tests. Please note, only the core tests were tracked and linked against requirements. Each core test has the identified description, the referencing issue number and assigned individual running and validating the test completes/fails.

Team members also tested using the black-box methodology, the official project **User Manual**.

Table 1 - Github issues list and the linkage to requirements

(https://github.com/jnaithani/genetic-programming/issues?state=open)

Test Issue Number (#)	Associated Issues (#)	Requirement
#20	#2	Mutation
#21	#6	Crossover
#23	#14	Selection evaluation
#22	#9	Fitness evaluation
#24	#15	15 Minute exit program

Lessons Learned

- **Time management.**
  - Time management was challenging considering that all the team members not only came from diverse background but some of us were full time employed.

- **Use case coverage estimations are inadequate.**
    - Always felt that the use case coverage estimations were inadequate. We were always concerned about the coverage and how are we going to test a specific use case.

- **Bug fix estimations go wrong.**
    - Whenever we were encountering a bug, it used to take more time than actual estimations. Re-testing the test case with the bug fix was always challenging.

- **Balanced participation among team members.**
    - Most of our team was full time employed. It was a challenge to get balanced participation.

- **A documented process does not guarantee the process will be followed.**
    - We had lot of documentation as to how we will achieve the goal. The documentation included details DFDs, Class diagrams, Object Diagrams, Sequence and Interaction Diagrams. Following all documented process was a challenge to us.
- **Get your priorities right.**
    - Implement essential features before playing with the more interesting, but expendable nice-to-haves.

- **Write reusable, abstract code, but don't go over the top. Code should survive minor specification changes, but it doesn't have to withstand a complete overturning of the project's purpose.**
    - Writing modular and reusable, abstract code was a challenge.
    - Finally the system developed was able to survive the changed specifications.

- **Naming user interface that is simple and understood easily.**
    - Challenge was to not use technical terms in user interfaces. Normal people don't understand them. We were prepared to name interface elements in a way that seem logical to all.

- **Code documentation well.**
    - Documented the code *extremely* well. Took time to write all those JavaDoc tags and keep them up to date. When creating a new class or method, documented those before we actually implemented them.

Project Summary

The current GP System implements the software development phases and techniques.

- At each phase and techniques, the software development object-oriented approaches were followed.
- A project plan document reflected details of all phases including Analysis, Design, Testing, and SCM.
- Focus was on Input, Training Data, Settings, Output, Display and Data Capture, Selection, Crossover, Mutation, Time Constraint
- Modeling and Preliminary Design
    i. Data Flow Diagrams
    ii. Class Diagrams
    iii. Interaction Sequence Diagrams

The GP System is designed and developed using object-oriented approach.

- Followed all iterative/incremental methodology.
- Focus was on objects and reusability.
- Software Architecture included:
    i. Java technology
    ii. Object Oriented approach
    iii. Modular Design approach
    iv. Simple User Interface – low priority
- Data Structure
    i. Binary Trees and Nodes
    ii. Program Data Structures – custom developed
        - Training Data
        - Settings

- Functions – utility methods
    i. Random number generators
    ii. Genetic Operators

The system provides excellent Tractability between all activities in its documents.

- Documentation tractability made easier. Used the following tools in the GP system development:
- Weekly on-campus and online team meetings
- Use of online tools
    i. Google Docs
    ii. GitHub
    iii. TeamViewer
    iv. Skype
    v. LucidChart
    vi. Google IO
    vii. Microsoft Office
- Development Tools
    i. Eclipse
    ii. IntelliJ

This system is flexible enough to accommodate the changing user requirements.

- The current Genetic Programming System was able to survive minor change specifications:
    i. Original target function was: $y = (X^2 + 1) / 2$

- **New Requirements:**

i. New target function was: $y = (-3X^2 + 7) / 2$

ii. Optional target function was: $y = (-3X^3 + 7) / 2 , \quad -2 <= x <= 2$

Post Project Analysis

PMA Definition:

The Post-Mortem Analysis can be summarized as:

- The process of looking back at a completed project's design and its development process, in order to identify those aspects where improvements can be made in future projects
- Post-mortems enable individual learning to be converted into team and organizational learning.

1. **What did we do?**
   - **a)** Conducted a PMA after our project presentation is done.

2. **What could have done better to improve the functionality to reduce issues encountered during development?**
   - **a)** Error prevention Vs. Handling errors after it occurred.
   - **b)** Test plan improvements.
   - **c)** Number of test cases that focuses on quality.
   - **d)** GUI?
   - **e)** More Modularity to include a balanced approach.

3. **When was PMA performed?**
   - **a)** PMA is ideally performed either soon after the most important milestones and events or at the end of a project, both in successful and unsuccessful software development projects.
   - **b)** The benefit is that post-mortems can often reveal findings more frequently and differently than project completion reports alone.

   We Conducted a PMA after our project presentation is done.

4. **What did we benefit for performing PMA?**
   - **a)** Helped project team members share and understand each other’s perspectives.

b) Integrated individual and team learning and how to improve upon.

c) Identified hidden problems and every team member go an understanding of how we handled major defects encountered during execution:

- Index Out Of Bounds Exception
- Divide by 0
- Stack Over Flow errors in computing Tree size()

d) Documented good practices and problems (so as not to repeat bad practices).

e) Increased job satisfaction by giving people feedback about their work.

f) Did lessons learned session.

g) Did project summary session.

5. What was our team AIM to do with PMA?

The Genetic Programming System aim of any post-mortem was to provide answers to the following four key questions:

a) What did we do well that does not have to be further discussed (the positive aspects)?

b) What did we learn (the negative aspects)?

c) What should we do differently the next time (the negative aspects which require improvements)?

d) What still puzzles us (share the things that are confusing or do not make sense)?

6. How did we conduct PMA?

The PMA process for average-sized and large projects is defined in five steps:

a) Plan a project review

b) Gather project data

c) Hold a post-iteration workshop or post-mortem review

d) Analyze the findings and synthesize the lessons learned

e) Publication of the results

7. What could have caused our project failure?

The following points could have caused failures if:

a) We don't look back until the very end of the project.

b) We don't set a tone of open and honest feedback.

c) We don't look at the whole picture of product and process.

d) We don't actually follow through on the feedback.

8. More Modularity to include a balanced approach?

We did good without going overboard with UI details.

9. What did not work well?

Effectively using the combined abilities of the team to handle parts of the project. Had the developers provide directions to the rest of the team on how to execute the program - they could have performed black box and functional testing while the development was on going. Also, they could have got a headstart on the test plan and procedures. Much of this was done after the fact - and when our demo was already done.

10. What worked well?

Using GitHub, Skype, TeamViewer, email effectively.

Project plan - We did a good job of the project plan submitted prior to midterm. Was complete and detailed.

Worked well together on the presentation and group assignment. Did well on our class presentation (that is what the Professor said as well).

DFD Diagrams - this was used by me heavily in developing components of the system. The DFD was attached to all issues to describe which part of the DFD was being worked on, and it stayed true to the high level context.

11. What could have worked better?

Design - although we did initial class diagrams and interaction diagrams, we did not use them in its entirety during development. Not sure if needed to either for this project - we had a good idea how to develop it from our discussions and that is what we did for our final project.

Testing - We could have done a better job of developing the tests and performing the tests.

Metrics and Measurement - Capturing the metrics was one thing, but effective analysis of the data was lacking. We did not use the information from our metrics to make further improvements.

Time Management - we all should have started earlier. Most of the activity on the project has occurred in the last two weeks - which is quite unbalanced. Also having full time jobs was not always problem for all of us. It might have been in determining meeting times during the week. Also being full time students provides time management challenges.

12. Summary

Overall, each team member was satisfied with the outcome of the project success as it not only adapted all the software development phases and techniques but was modular, tractable and this Genetic Programming System is flexible enough to accommodate the changing user requirements.

A project well done!

References

- **How the referenced materials were used in the Genetic Programming System**
  - **For developing the GP System use of information available at the following locations and reference text books**
    - http://www.genetic-programming.com/gpquadraticexample.html
    - http://www.geneticprogramming.com/Tutorial/
    - http://geneticprogramming.us/Selection.html
    - http://docs.oracle.com/javase/tutorial/collections/algorithms/#sorting**5.**http://books.google.com/books?id=Bhtxo60BV0EC&pg=PA82&lpg=PA82&dq=The+protected+division+function+%25+returns+a+value+of+1+when+division+by+0+is+attempted+genetic+programming&source=bl&ots=9oiRjwg-MN&sig=y5LB1tRPjWwtorVGVgg6Gdi9c1Q&hl=en&sa=X&ei=WiiKUo7zJe3lsASf8YHYCA&ved=0CDYQ6AEwAQ#v=onepage&q=The%20protected%20division%20function%20%25%20returns%20a%20value%20of%201%20when%20division%20by%200%20is%20attempted%20genetic%20programming&f= false
    - http://www.amazon.com/Genetic-Programming-Introduction-Artificial-Intelligence/dp/155860510X/ref=sr_1_3?s=books&ie=UTF8&qid=1385953621&sr=1-3&keywords=Genetic+Programming
  - **For testing the results of the program and simplifying the final solutions algebraic expression**
    - http://rechneronline.de/function-graphs/
    - http://www.wolframalpha.com/widgets/view.jsp?id=aeecc5a9c646444f00978ed43e747a96
    - http://www.wolframalpha.com/widgets/view.jsp?id=97ffcbd95363387c7e371563057eb02f

Weekly SCM

SCM Template

- Name of work
- Revision date
- Changes
- How
- Where

Week 3:

Revision Date	Who	Name of Work	Changes	How ?	Where ?
09/22/2013	Warsame Bashir	Problem Description	Draft	Online Document	Google Docs
09/22/2013	Jayesh Naithani	GitHub Repository	Initial	Online Document	GitHub
09/24/2013	Team	Work Break Down	Collectively made a task list.	Meeting	Lab

Week 4:

Revision Date	Who	Name of Work	Changes	How ?	Where ?
09/29/2013	Warsame Bashir	SRS - Template	Initial release of Software Release and requirements template	Online Document	Google Docs

Week 5:

Revision Date	Who	Name of Work	Changes	How ?	Where ?
10/06/2013	Team	Requirement Analysis	Initial draft of the requirements analysis file.	Online Meeting	Google Docs & Google Plus
10/06/2013	Team	Requirement Analysis	Identified the functional and behavioral requirements .	Online Meeting	Google Docs & Google Plus
10/06/2013	Team	Requirement Analysis	Determined which analysis and design document were needed before implementation.	Online Meeting	Google Docs & Google Plus

Week 6:

Revision Date	Who	Name of Work	Changes	How ?	Where ?
10/12/13	Rajesh	UML	Initial design	Meeting	Google Docs - Draw.io
10/12/13	Kholoud	UML	Initial design	Meeting	Microsoft Word
10/08/13	Warsame	Code Commit -	Added settings.java, GP_Main.java , Tree.java	Meeting	GitHub
10/09/13	Jayesh	DFD	Initial Level - 0	Meeting	Lucid Chart
10/12/13	Jayesh	DFD	Level - 1 Diagram	Meeting	Lucid Chart
10/13/13	Team	Requirements Analysis	Initial Design - Domain model diagram, testing plan, sequence diagrams.	Meeting	WhiteBoard - draw.io
10/13/13	Team	Project Milestone	Initial	Meeting	Excel
10/15/13	Rajesh	UML	OO Diagrams complete	Online Document	Google Docs - Draw.io
10/15/13	Team	Project Plan	Revision and Changes to final document complete	Online Meeting	Google Docs & Google Plus

Week 7:

Revision Date	Who	Name of Work	Changes	How ?	Where ?
10/17/13	Jayesh	Data Structure (Trees)	Initial data structure of nodes, trees and operations	Task/Online	Eclipse and Java
10/23/12	Warsame	Mutation	Mutator function to change a binary node's value	Task/Online	Eclipse and Java
10/17/13	Rajesh	Environment Setup	Setup the development environment	Task	Eclipse and Java
10/23/13	Rajesh	Main	Write Main program	Task	Eclipse and Java
10/25/13	Rajesh	Testing	Testing available functions	Task	Eclipse and Java

Week 8:

Revision Date	Who	Name of Work	Changes	How ?	Where ?
10/30/13	Jayesh	Crossover Selection	Added the crossover and selection methods.	Task/Online	Eclipse and Java
11/08/13	Rajesh	Testing	Testing available	Task	Eclipse and Java

			functions		
11/08/13	Rajesh	Project Presentation	Create and Work on Project Presentation	Task/Online	Eclipse and Java, PPT
11/19/13	Team	Project Presentation	Review/Edits to Project Presentation	Task/Online	Eclipse and Java, PPT

Week 9:

	Who	Name of Work	Changes	How ?	Where ?
11/19/13	Jayesh/Warsame	Crossover Selection, Mutation	Added the crossover and selection methods.	Task/Online	Eclipse and Java
11/24/13	All	Main Program, Testing	Finalizing Maina Program,Testing available functions	Task	Eclipse and Java
11/26/13	Rajesh	Project Presentation	Added details to the Project Presentation	Task/Online	PPT
11/30/13	Team	Project Presentation	Review/Edits to Project Presentation	Task/Online	Eclipse and Java, PPT

Week 10:

	Who	Name of Work	Changes	How ?	Where ?
12/4/13	All	FInalize Project Report	Metrics, Project Summary, Lessons Learned, User Manual	Task/Online	Google Docs
12/7/13	Warsame	Test Plan	Documentation, Additional Test Cases	Task	Eclipse and Java
12/10/13	All	Final Project Report	Consolidated all documents	Task/Online	Google Docs

			into single project report		

User Manual

Pre-Installation

Before you begin the install process for the Genetic Programming System, verify that the following application requirements are met:

- A Java 1.6 Java Runtime Engine (JRE)/Java Development Kit (JDK) or greater is installed on your desktop or laptop. If you don't have the JRE/JDK installed, it can be downloaded and installed from:

  http://www.oracle.com/technetwork/java/javase/downloads/index.html

- Ensure that the path to the JRE/JDK bin directory has been added to the environment execution path. You should be check this by performing the following command:

  ```
  java -version
  ```

  If the JRE/JDK bin directory is in the environment path, then the version of the JDK should be printed. For e.g,,

  ```
  java version "1.7.0_01"
  ```

Installation

Following are the installation procedures:

- Download the installation file
  - Create a directory on your local file system
  - Using a web browser, download the **`gps.zip`** file into this directory from the the following location:

    https://github.com/jnaithani/genetic-programming/tree/master/deliv
- Extract files into local directory
  - Open a command line window on your desktop or laptop

- Extract files from the the **gps.zip** file by executing the following command from the command line:

```
jar xvf gps.zip
```

- Verify that the following files were extracted into the current directory
  - gps.jar
  - settings.properties
  - trainingdata.txt
  - run.bat

Settings

The Genetic Programming System settings are configured via the following file:

settings.properties

The file is preconfigured with the following settings:

```
## Genetic Programming System Settings

#Percent of population that should undergo mutation.  The
percent range is between 0 and 1.
mutatorprobability=0.2

#Percent of population that should be undergo crossover.  The
percent range is between 0 and 1.
crossoverprobability=0.8

#Percent of current population, the most fit, that should be
selected unchanged for the next generation. The percent range is
between 0 and 1 and should be kept at 1 - crossoverprobability.
survivalprobability=0.2

#Acceptable fitness level, or fitness threshold, of final
solution.
fitnessthreshold=0.1

#Max time limit in milliseconds=15 x 60 x 1000
maxtimelimit=900000
```

```
#Initial population size
populationsize=100

#Max number of generations before regenerating all the trees
maxgenerationcount=1000

#Max depth of a GP tree.  The depth is the count of the number
of nodes, not edges, in the longest path.
maxdepth=4

#Max depth limit to avoid excessive growth of trees after
crossover
maxdepthlimit=8
```

Training Data

The training data delivered with the Genetic Programming System is located in the following file:

```
trainingdata.txt
```

It is a comma separated value (CSV) list of (x,y) values for the target function:

$$(-3x^2 + 7)/2, \text{ for } -10 <= x <= 10$$

Running the Genetic Programming System

To run the Genetic Programming system do the following:

- Open a command line window and changed directory to where the Genetic Programming System is installed
- Run the program by executing the following script:

  run

The program will immediately begin executing and display results as it is executing on a graphical display and on the command line.

The background text area on the graphical display will become green once the program has completed and displayed the final results.

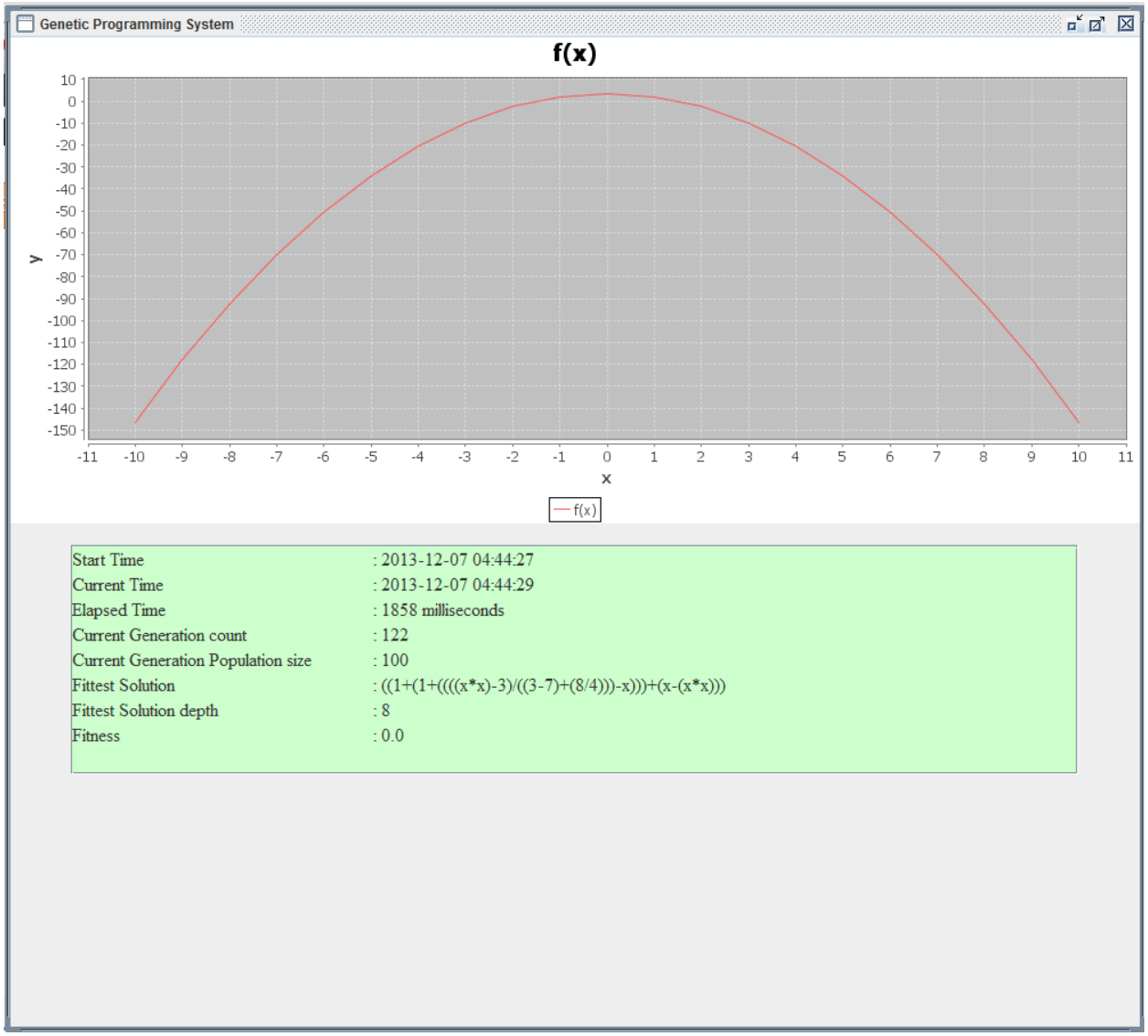
To exit the program, close the graphical display.

Repeat the above steps to re-run the program.

Output Display

The results of the program while it's executing is displayed on the graphical display and well as on the command line. Once a solution has been found or the max execution time limit has been exceeded, the program displays the final test results in the text area and the background is displayed in green.

The graphical display plots an XY graph of the best solution in each generation as it is executing, including the XY graph of the final solution.



The display text area aboves show the following information:

- Start Time - the time the program was started
- Current time - the current program execution time
- The total elapsed time in milliseconds
- The current population generation count
- The current population size
- The expression for the fittest tree in the current generation population
- The depth of the fittest tree in the current generation population
- The fitness value of the current fittest tree in the current generation population

To redirect the output in the command to a file called “output”, do the following:

```
run > output
```

Output Charts

The following chart data is also recorded and stored in the same directory when the Genetic Programming System files are installed.

Chart: initial_population_fitness.jpg

This chart records the initial population fitness values.

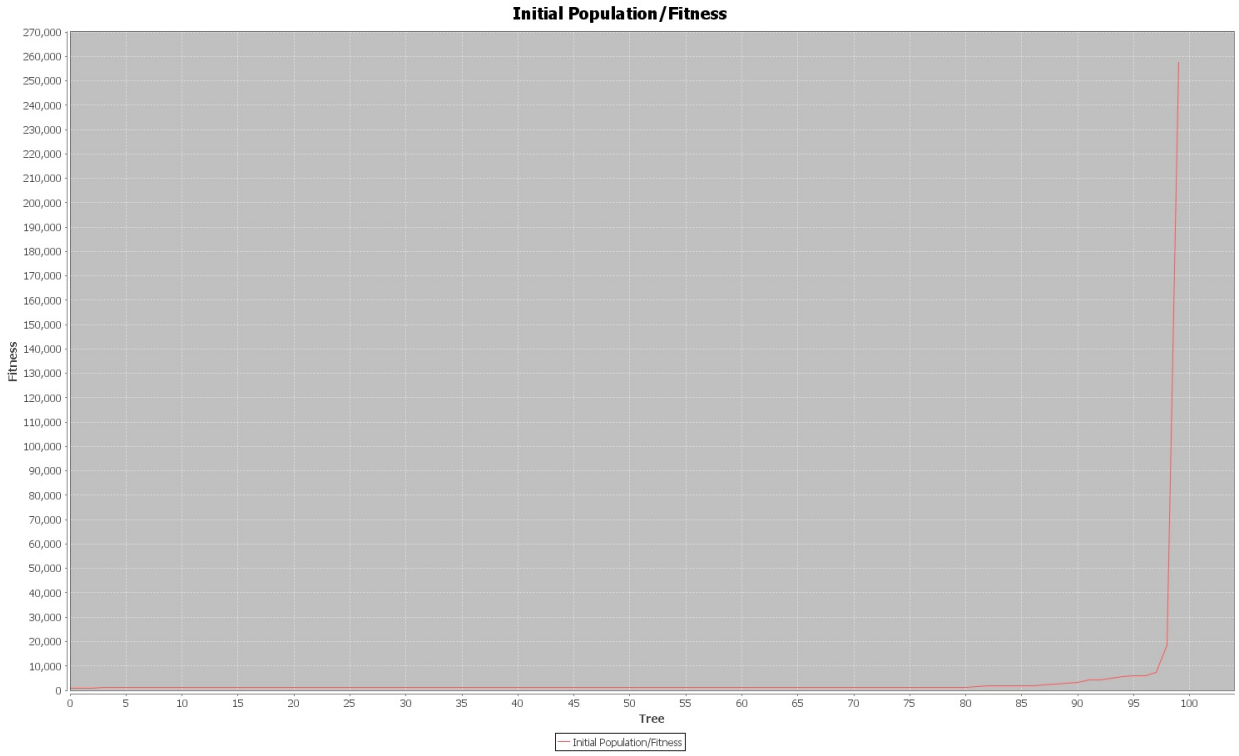


Chart: final_population_fitness.jpg

This chart records the final population fitness values.

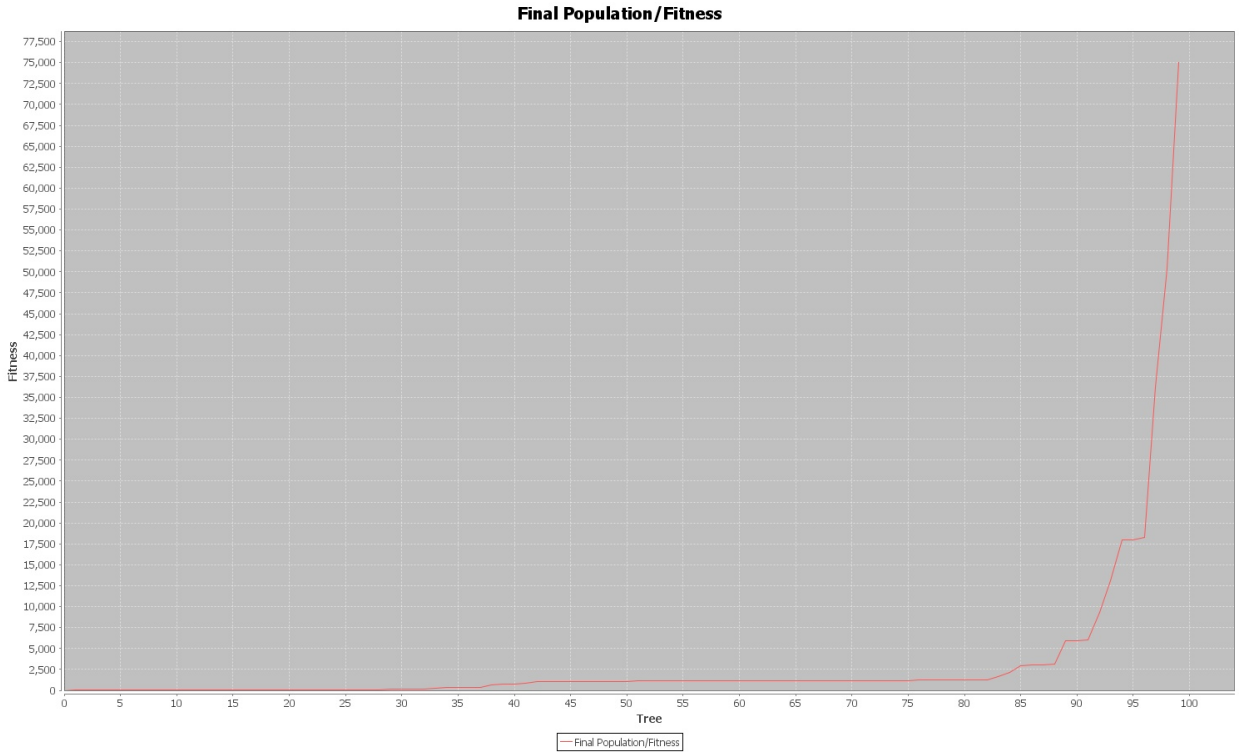


Chart: bestfitness_generation.jpg

This chart records fitness value of fittest tree in each generation.

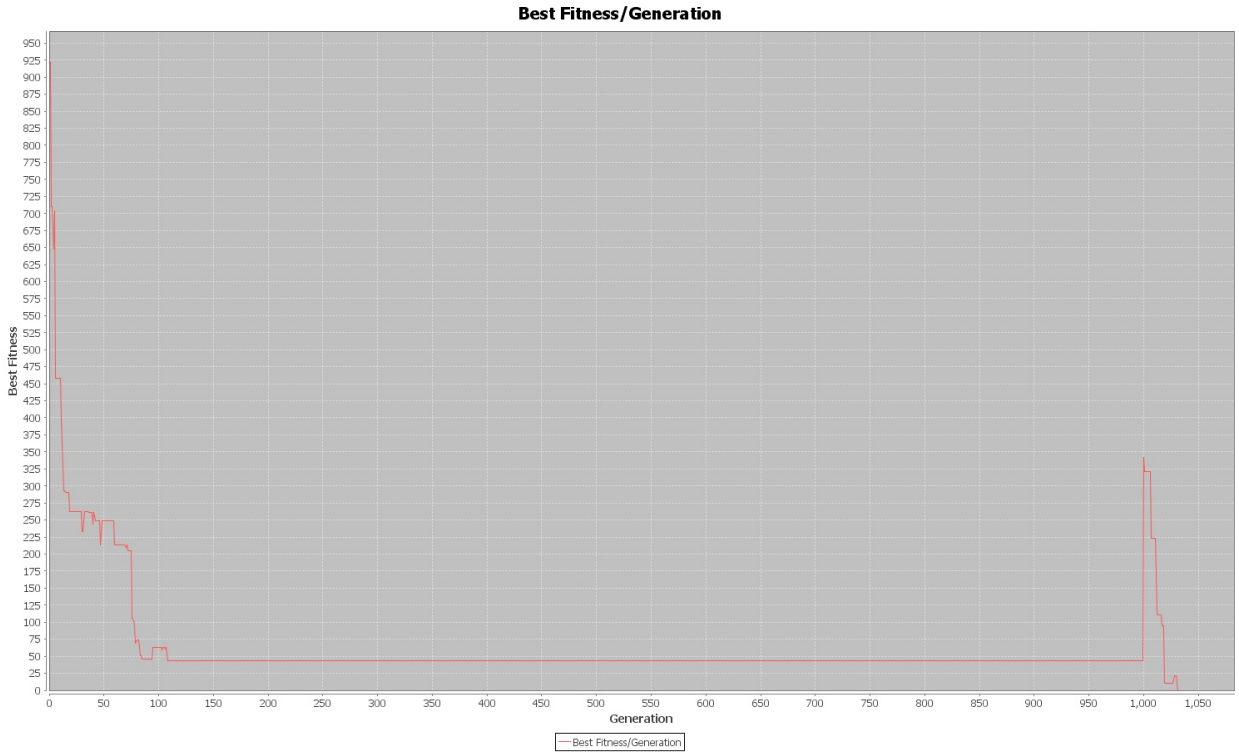


Chart: xy_graph.jpg

This chart plots the XY values for the fittest solution found by the program in the end.