

easyGP: Genetic Programming as a Service

Pranjal Chakraborty
Department of Computer Science
Brock University
Ontario, Canada

1. INTRODUCTION

Setting up the most basic GP system can be a daunting task, more so if the user needs it quickly and is trying to use ECJ [7] for the job. ECJ needs a lot of initial setup steps to have a project running, and it is very easy to commit mistakes in the parameters file, particularly when we are defining functions. This project explores the possibility of creating an out-of-the-box GP system, that can be deployed easily, and can be used conveniently without going through all the trouble of creating functions and nodes specific to the task, and creating a fitting parameters file. In this project, we will be building a system, where we will have a web interface in the front, where the user will interact with the system, and in the backend, as GP service (leveraging ECJ) will deal with the task that the user has submitted and send the results back to the frontend.

2. SYSTEM DESIGN

2.1 Defining the Features and the Scope

To effectively design our system, we need to define what our system features will look like and what will be the scope. There will also be some assumptions to make our design decisions more concrete. In this section, by "user", we mean the end user, by "developer", we mean the person who will be deploying and maintaining the system. User will be interacting with the system only through the UI, and the developer will be interacting with the system through source code.

1. There is no user management component in the project. Everyone will be able to see everyone's data, and there will not be any way to differentiate which information is managed / created by whom.
2. User will be able to upload a dataset as a CSV (Comma-separated values) file. Files with or without column headers are acceptable (an appropriate option in the UI will be there). The assumed properties of the uploaded file are-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2022 ACM X-XXXXX-XX-3/18/22 ...\$15.00.

- File size is within 10 MB.
- Each row corresponds to a separate data, and is comma delimited.
- For each row, the last component is the dependent variable and can be both number (regression) and text (classification). All the previous components in each row is a number.
- There are not more than 30 independent variables.
- There are no inconsistency in the independent variables count among the rows.
- The file only contains UTF-8 text.

3. User will be shown a list of choices of predefined GP functions. User must select at least 1 of these functions. The predefined GP functions are-

Table 1: GP Function Options

Function	# of children
Add	2
Sub	2
Mul	2
Div	2
Sin	1
Cos	1
Max	2
Min	2

4. The system can deal with Symbolic Regression problems and Classification problems. The user will be shown these 2 options and they will choose the type of problem. It is assumed to be consistent with the data that they had uploaded.
5. User will have the option to change some GP parameters. It is assumed that the user understands the nuances of these parameter values. If the user does not edit a particular value, a default value for that parameter will be chosen under the hood. The available parameter choices are-

Table 2: GP Parameter Options

Parameter	Default Value
Population	512
Crossover Rate	0.9
Mutation Rate	0.1
Generations	51
Tournament Size	1
Elitism	0
Jobs	5

6. User will be able to see the list of all the tasks that had been added so far in the system and their status. They can select a task from the list to see the details. Alongside regular details (e.g. dataset file name, uploaded date etc.), a task with "Completed" status will show the tree with the maximum fitness among all the runs, and a graph with the best run and average run.

2.2 Architecture

The complete system consists of 3 components, including 2 services and 1 database (Figure 1).

2.2.1 Database

For our database, we have chosen to use MySQL. There, we have created a database called **gpproject** and will be using it as our storage. For entity creation and management, we are using ORM management tools in our services side, so we will not be creating any tables manually in our database.

We will be having 3 tables in our database- Config, Dataset, Result. To have these entities loosely coupled for now, we did not put any explicit mapping between them through ORM, but there is an underlying one-to-one mapping between them through **uuid**, and they can be referenced from each other by this value. Absence of explicit mapping may cause orphan entities (e.g. user uploads a dataset, but leaves without creating the GP configuration in the next page), but it will not break any feature.

2.2.2 Frontend Service

This is not a traditional frontend service. Apart from interacting with the user, it will also do the data pre-processing and constructing JSON formatted data, before dumping the data into the database. The technical details of this service is as follows-

- Design Pattern: MVC (Model, View, Controller) [6]
- Web framework: Spring [3] Boot
- View processor: JSP (Jakarta Server Pages)
- ORM (Object-Relational Mapping) Tool: JPA (Jakarta Persistence API) with Hibernate implementation.
- Server: Tomcat (Embedded)

This service is responsible for user interaction, and it will be able to handle concurrent users. It has one controller to manage all the routes. The available controller routes are listed in Table 3.

Our frontend service will be producing **Config** and **Dataset** entities from user inputs for our database. The GP configurations that the user has chosen, are modeled into **Config-Model** object and converted into JSON string, to be wrapped

with **Config** entity and stored into the database. Similarly, user uploaded dataset is modeled into **DatasetModel**, converted to JSON, wrapped with **Dataset** entity and stored. This intermediary JSON marshalling / unmarshalling creates an overhead, but makes it very convenient to communicate within the services.

For view styling, we have used a couple of third party libraries.

- **Bootstrap** [1] has been used to make the view responsive.
- **jQuery** [2] is used for easier HTML manipulation.
- **DataTables** [5] has been used for tabular view of the tasks.
- **Chart.js** [4] is used for plotting the graphs.

2.2.3 Backend Service

As a backend service, we have a singleton ECJ instance, which will only execute one GP task at a time. GP tasks are computationally expensive, and we can easily overwhelm a system by having multiple ECJ instances running in the background. To prevent that, by this design, this service will periodically look into the database for new tasks, take one task at a time and after finishing the task will look for another one.

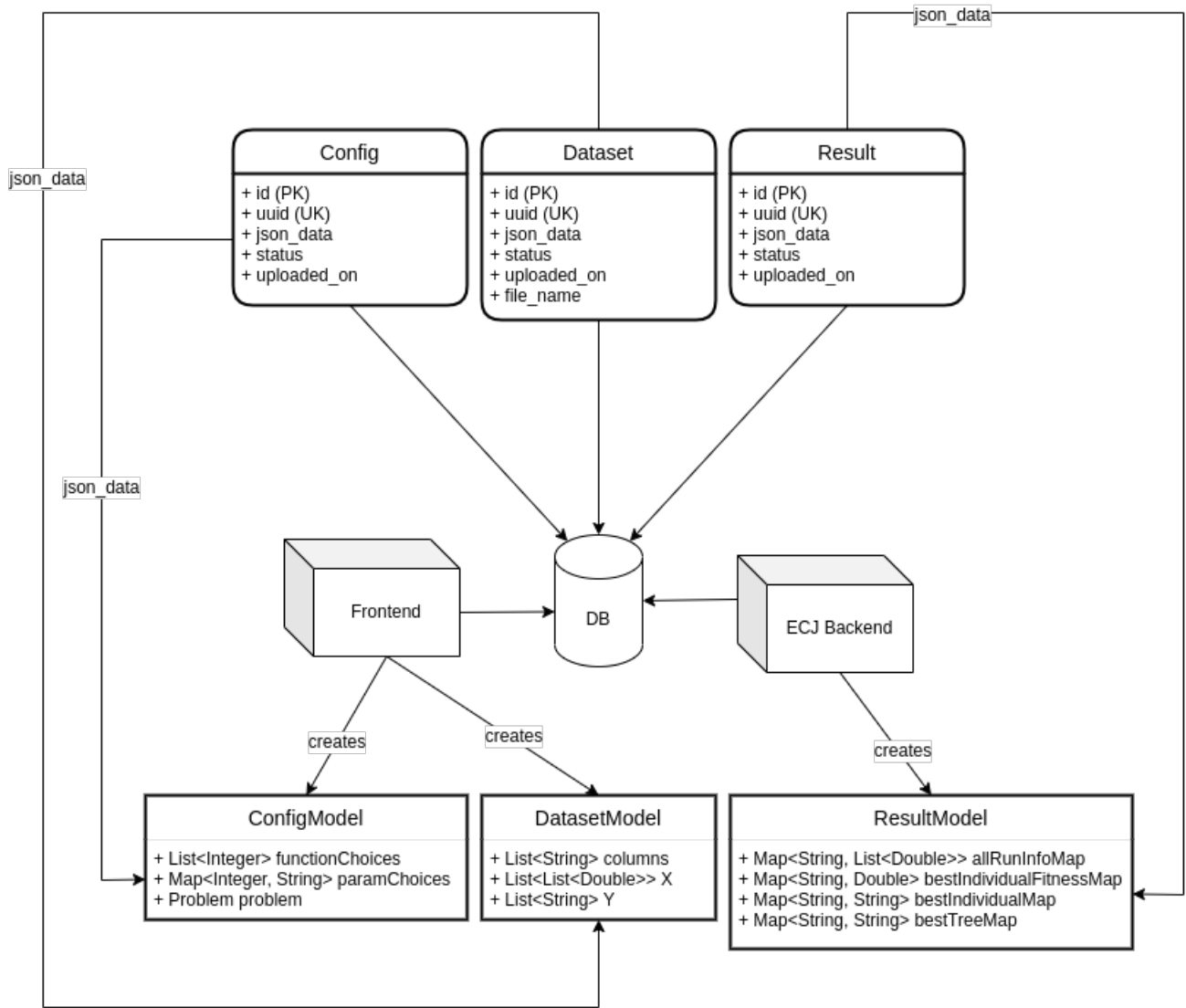
To keep our background service flexible to different types of data, we needed to enhance the interaction with ECJ.

1. **Predefined nodes:** In ECJ, we define separate nodes for each of the independent variables. Each of these nodes have their own class files, which is not possible to be created dynamically based on the dataset. To tackle that, we decided that a maximum number of 30 independent variables will be handled by our system. Based on the user data, we will be using only the amount of nodes that we will need.
2. **Predefined functions:** We also implemented a couple of most used GP functions (table 1), ready to be selected by the user for their GP system.
3. **Dynamic RunConfig:** When we try to design to solve a particular GP problem with a particular format of dataset using ECJ, we define all the moving parts (functions, nodes, GP parameters) of our problem in the **.params** file. For our case, most of the params will come from the user input. To handle that, we exploited the built-in feature of ECJ to read params as command line arguments. We generated this dynamic command line arguments based on the user input, keeping our params file only a couple lines long.
4. **A proxy Evolve class:** The default **Evolve#run** method of ECJ performs a system exit at the end of the run, which does not conform to our design to perform GP tasks one after another in a loop. For that we have copied the elements of the default **Evolve** class and made a proxy one, without the system exit.

Our backend service will be producing **Result** entities, and will be consuming **Config** and **Dataset** entities. Just like our frontend service, we are using JPA (Hibernate implementation) to manage our database entities.

Table 3: Controller Routes

Route	HTTP Method	Responsibility
/	GET	View, with the list of tasks. Has button to add new task.
/upload	GET	View, to upload dataset.
/upload	POST	Handles user uploaded data, stores into the database, and redirects to <code>/params</code> .
/params	GET	View, to submit GP params (functions, parameter values, problem type).
/params	POST	Handles user submitted parameter configurations, stores into the database, and redirects to / (home).
/task	GET	Show details of a task and results (if completed).
/error	GET	Generic 404 error page.

**Figure 1: The System Architecture.**

One of the places where we lost the flexibility is in the definition of fitness function. We could not come up with a strategy to read the fitness function from the user. For this reason, we have decided to implement the most basic functions for our supported problems.

Table 4: Fitness Functions

Problem	Fitness function
Symbolic Regression	SAE (Sum of Absolute Errors)
Binary Classification	Accuracy (hits / Total # of data)

We have defined the thread counts based on our system.

Table 5: Predefined GP Params

Parameter	Value
Eval Threads	6
Breed Threads	6

2.3 Developer Guide

The project was developed and tested in a machine with following specifications-

- OS: Linux Mint 20.3 Cinnamon
- Processor: AMD Ryzen 5 3600 6-Core Processor x 6
- Memory: 31.4 GiB

To successfully deploy the system locally, one is assumed to have the following tools installed-

1. JDK 11 or newer
2. Gradle 7.4
3. MySQL 8.0.28
4. Git 2.25.1

2.3.1 Database

1. Create a database named `gpproject`. (If any other name is chosen, the database properties in the services need to be updated accordingly).
2. The database user (in our project, 'root') might need full access for the database to be accessed from other services. In that case, the user needs to be granted full access. For our project, we executed the following command in MySQL.

```
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' WITH GRANT OPTION;
```

2.3.2 Frontend Service

1. Clone the project by executing this command- `git clone git@bitbucket.org:pcborty/ecjfrontend.git`
2. Build the project. `gradle clean build`
3. Run the `ECJFrontendApplication#main` method inside the project.

2.3.3 Backend Service

1. Clone the project by executing this command- `git clone git@bitbucket.org:pcborty/ecjbackend.git`
2. Build the project. `gradle clean build`
3. Run the `Main#main` method inside the project in a different terminal window.

The services can be deployed in any order. The database name, username, and password can be updated for the services through the `application.properties` file in the frontend service, and the `persistence.xml` file in the backend service.

2.3.4 Adding Features

- To add a function -
 1. Backend service: Add the new GP function into the `gp.functions` package. Then add an entry to the `db.utils.Util.FUNCTION_CHOICES`.
 2. Frontend service: Add an entry to the `ecjfrontend.helper.utility.FUNCTION_CHOICES`.
- To add an editable GP param -
 1. Backend service: Add an entry to the `db.utils.Util.GP_PARAM_CHOICES`.
 2. Frontend service: Add an entry to the `GP_PARAM_CHOICES` and `GP_PARAM_DEFAULT_VALUES` in the `ecjfrontend.helper.utility.Utility`.
- To add a problem -
 1. Backend service: Add the problem in `gp.problems` and add an enum entry to the `db.models.Problem`.
 2. Frontend service: Add an enum entry to the `ecjfrontend.domain.Problem` and insert an entry into `PROBLEM_CHOICES` of the `Utility` class.

3. USER GUIDE

3.1 Home page

When the user visits the webpage directly (/ route from table 3), they are greeted with a home page (figure 3), with a list of tasks that have been created so far.

- They can choose how many tasks they want to see by selecting the number of entries.
- They can search any particular task with any full or partial information that they can remember in the table, using the search bar.
- Furthermore, they can sort the tasks by columns alphabetically by clicking on the column headers.

At the bottom of the table, there is an "Add Task" button, clicking on which will redirect the user to a page to add a dataset.

3.2 Add / Upload Dataset

In this page (figure 4), user will be uploading their dataset. If the dataset has column headers, user must select the checkbox. If there are issues with the file format or data, there will be validation errors, shown within the view, with an opportunity to re-upload.

Clicking the "Upload" button will take the user to the parameter configuration page.

3.3 Configure GP Parameters

In this page (figure 5), user will be playing with the GP parameters.

- From the "Available Function" list, user can choose which GP functions they want to integrate into their GP task. Only the selected functions will be used.
- From the "Choose your problem" option, user can choose whether they are trying to solve a binary classification problem, or a symbolic regression problem.
- User can edit their preferred values for the listed GP params into the fields on the right side of the page. There are validations for text values or negative values in these fields, and user will be able to correct the mistakes.

3.4 Seeing the Result

From the home page, user can click on a particular task, and they will be redirected to this page (figure 6). If the status of the task is "Completed", user will be able to see the results of the run, alongside some dataset related information, that are available regardless of the status.

In the results of a GP task (with the status "Completed"), user will be able to see the GP tree of the best performing individual (figure 7) and a graph (figure 8), plotted with the best run and the average run in terms of fitness. User will be shown a button to download this plot.

4. WISCONSIN BREAST CANCER DIAGNOSTIC DATA SET: A SAMPLE RUN

In this dataset, after omitting the ID column, there are 30 independent variables and 1 dependent variable. This is a binary classification problem. To prepare the data for easyGP, we moved the dependent variable as our last column, removed the ID column, and constructed a CSV file out of it. Then, in the easyGP interface, we clicked the "Add Task" button. After that, we selected the file, unchecked the box (as we did not keep any column headers) and clicked "Upload".

In the GP configuration page, we selected all the functions and chose "Binary Classification" from the problem selection section. Then we chose the following (table 6) GP param values.

After clicking "Submit", the status of the task was "Pending" for a while, and changed to "Processing" a bit later.

After about 5 minutes, the status was changed to "Completed". The plotted graph looked like figure 2-

Table 6: GP Parameter Options

Parameter	Value
Population	2048
Crossover Rate	0.9
Mutation Rate	0.1
Generations	100
Tournament Size	3
Elitism	0
Jobs	10

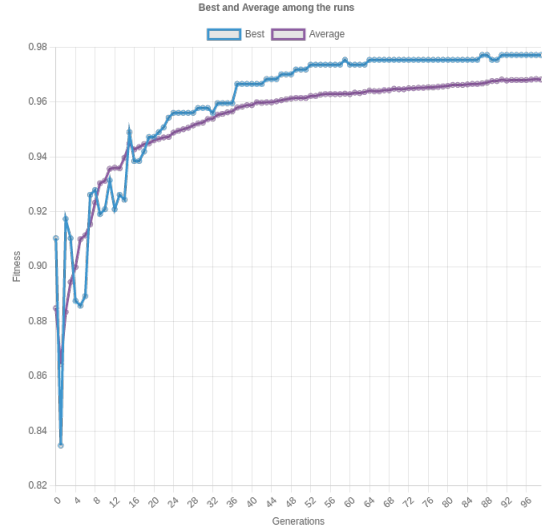


Figure 2: The Generation vs. Fitness Graph

The best performing GP tree among all the runs looked like this-

```
(- (- (min (- (+ x12 (* (max x8 x25) x24)) x3) (min
x4 x24)) (/ (- (max (- (+ x12 (* (max x8 x25) x24))
x3) (+ x14 (* (* x4 x13) (/ (/ x15 x12) (cos x13))))))
(- (max (- x22 (+ (- x30 x19) (max (max x19 x19) (max
x29 x4)))) (* (min (- (min x16 x25) (- (+ x3 x9) (*
x7 x24))) (* x26 x2) (+ x4 x19))) (max (max (max (*
x7 x24) x23) (+ x12 (- x10 x7))) (min (/ (/ x15 x12)
(* x4 x13)) (- x7 x1)))) (/ (- x7 (- x29 (max (+ x14
x3) (cos x26)))) (cos x27)))) (/ (max (sin (* (min (-
(/ (- (* x13 x30) (/ x12 x11)) (cos x27)) (max (cos
x1) x26)) x5) (sin (* (max x8 x25) x24)))) (cos (cos
x29))) (/ (max (max (max (max (cos (+ x12 x7)) x2) (max
(cos (+ x12 x7)) (max x8 x25))) (min x16 x25)) (+ x24
(/ (- x10 x23) x27))) (max (* x23 x23) x23))))
```

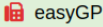
5. CONCLUSION

In this work, we explored how we can design and implement an end-to-end GP system in the context of SaaS (software as a service). We have tackled 2 fundamental problem types- binary classification and regression. The next reasonable step would be to add feature to upload test data and see results on test data. There are also scopes for adding features to deal with multi-class classification and computer vision problems. For the computer vision problem, we will need to have new interfaces, as the type of data does not go with our current design. To summarize, there are a lot of scopes to improve upon this project, and we will continue to

add features to this work.

6. REFERENCES

- [1] Bootstrap. Available for free at <https://www.bootstrap.com/>.
- [2] jquery. Available for free at <https://www.jquery.com/>.
- [3] Spring. Available to use for free at <https://spring.io/>.
- [4] Chart.js, 2014. Available for free at <https://www.chartjs.org/>.
- [5] A. Jardine. Datatables, 2008. Available for free at <http://datatables.net/>.
- [6] G. E. Krasner and S. T. Pope. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49, aug 1988.
- [7] S. Luke. ECJ evolutionary computation library, 1998. Available for free at <http://cs.gmu.edu/~eclab/projects/ecj/>.



Show
10
entries

Search:

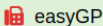
Created On	Task ID	Status
2022-04-23 08:50:51.485	5e1381ee-0785-4da6-b256-60c9f6a3537c	Completed
2022-04-23 08:29:49.481	da42595d-daf4-4f15-bf44-af968cebb9f3	Completed
2022-04-21 16:47:40.858	5add05b4-847b-4277-b5a7-f784a06c4e90	Completed
2022-04-21 15:17:24.842	1aaaa8e9-6f54-496b-a840-cd13533b8ed2	Completed
2022-04-21 15:11:17.983	84ddf532-176f-4c73-8d2a-2f90a861a344	Completed
2022-04-21 14:59:50.216	57559262-9442-42c1-bdf0-b0c34b5fdac2	Completed
2022-04-21 14:35:00.213	3ac3b564-1dd6-4275-9b6b-9bfce34bf4df	Completed
2022-04-21 14:23:24.532	245e8d21-e8fa-4682-87f7-f76f53dcd81b	Completed
2022-04-21 14:22:17.029	23c00c0e-92ac-4d05-9fee-5f7a7c266245	Failed
2022-04-21 14:19:04.598	2de46058-5214-450d-88e7-07d48f1019bf	Completed

Showing 1 to 10 of 22 entries

Previous
1
2
3
Next

Add Task

Figure 3: The Home Page.



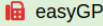
CSV File

No file chosen

Dataset contains column headers

☐

Figure 4: The Data Upload Page.



Available functions

☐ Add (Addition): 2

☐ Sub (Subtraction): 2

☐ Mul (Multiplication): 2

☐ Div (Safe Division): 2

☐ Sin (Sine): 1

☐ Cos (Cosine): 1

☐ Max (Minimum): 2

☐ Min (Maximum): 2

Choose your problem

☐ Binary Classification

☐ Symbolic Regression

Please choose your params with care

Population Size

Default = 512

Crossover

Default = 0.9

Mutation

Default = 0.1

Generations

Default = 51

Tournament Size

Default = 1

Elitism

Default = 0

Jobs

Default = 5

Submit

Figure 5: The GP Parameter Configuration Page.

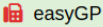
	
Task UUID	5e1381ee-0785-4da6-b256-60c9f6a3537c
Status	Completed
Problem	Binary Classification
Dataset file name	wdbc.csv
Task Created On	2022-04-23 08:50:51.485

Figure 6: The Result Page.

Best Individual Among All the Runs

```
(- (max (- (max (- (- (- (* x24 x18) (- x7 x3)) x18) (- (- (- (- (/
(max (- x28 x30) (cos (+ x5 x10))) x23) x3) (* x7 (sin x30))) (- x14 x3))
(- (max x26 x29) x3)) (min (- x8 (* x7 x24)) (- x17 x28)))) x17) (* (- (-
x17 x28) (- (max (* (+ (sin (- x17 x28)) x8) (- x17 x28)) (* (- (min (- x8
x6) x7) (- x17 x28)) (- x22 (- (- (cos (sin x17)) (- (* (sin x10) x24)
x3)) x3)))) (- x17 x28))) (- (* x24 x18) (- (max (* x14 x9) (- (- (cos
x13) (sin x13)) (cos (- (* x7 (sin x30)) x3)))) (min (- x8 (- x2 x12))
(min (- x8 (sin x30)) x18)))))) (max (/ (- (+ (sin x8) (- x6 x26)) x18) (/
(sin x3) (max (+ (cos (max x26 (- (* x7 x24) x3))) (max x26 x29)) (- (- (*
x24 x18) (sin x10)) x17)))) (max (+ (- (- (max (/ (max (min (- (/ x16 x29)
x3) (- (sin x30) (- x17 x28))) (max (- (min (- x8 x6) x7) (- x17 x28)) (*
(- (max x26 x29) (- (- x17 x28) x17)) (- (* x24 x18) x26)))) (- (* x3 x18)
(cos x16))) (* (- (- x6 x17) (- (* x7 (sin x30)) x6)) x12)) (min (max x30
(max (min (- (sin x30) (min (* x7 x24) x15)) (cos (sin (- x17 x28))))
x14)) (max (+ (/ x16 (max x26 x29)) (max x26 x29)) (- (- x8 (sin x30)) (-
x18 (- (max x26 x29) (- x17 x28)))))) (- (* x7 x24) x3)) (+ x8 (* (- (min
(- x8 x6) x7) (- x17 x28)) (- x22 (- (- (* x24 x18) (- x14 x3)) x3))))
(sin (- (* (- (- (- (min (- x8 (sin x30)) x20) (- x17 x28)) (- x17 x28))
(- x17 x28)) (- x17 (- (min (- x8 (- (min (- x8 x6) x7) (- x17 x28)) (max
(sin (- x17 x28)) (- x17 x28)) (- x17 x28)) x11)))) (* (- (min (* x17
x17) x16) (- x17 x28)) (- (* x24 x18) (- (max (- x17 x17) (* (- (min (- x8
(sin x30)) x20) (- x17 x28)) (- (* x24 x18) (- (* x7 x24) x3)))) (min (max
(cos (- x8 x6)) (max (- x28 (* (- x22 (- (- (cos (sin (- x8 x28)) (- (*
(sin x10) x24) x3)) x3)) (- x22 (- x17 x28)))) (sin (- x17 x28)))) (cos (-
x25 x17)))))) (max (/ (- (- x6 x17) (- x8 (sin x30)) (cos (/ (max (/ (-
(/ (max (max x30 (* (- (max x26 x29) (- (- (* x24 x18) (sin x30)) x28)) (-
(- x8 (- x17 (sin x8))) (- x7 x3)))) (* (- (- x17 x8) x12) (cos x17)))
x23) x18) (max (+ (- x22 (- (- (max (+ x17 (cos (- x2 x12))) x8) (- x17 (-
x17 x28)) x3)) (sin (- (* x7 x24) (cos x17)))) (- x22 (/ x16 (- x8
x6)))) (max (max (max (- x6 x17) (sin (- (- (sin (- (sin x13) (- x17
x28)) (- (min (- x8 (- x8 x6)) x20) (- x17 x28)) (/ x16 x17)))) x9) (-
(* (- (- (+ x17 x28) (- x17 x28)) (- x17 x28)) (- (sin x30) (- (min (- x8
(sin x30)) x20) (- x17 x28)))) (- (* x7 x24) x3)))) (- (- x1 x12) (- (*
x14 x9) (- x17 x28)))) (max (max (+ (cos (max x26 x29)) (sin (- (min (-
x8 (sin x30)) x3) (- x17 x28)))) (- (- (max (- x8 x6) (* (- (min (- x8
(sin x30)) (max (* (max (- x17 x28) x29) x9) x9)) (- x17 x28)) (- (- (*
x24 x18) (- (cos x13) (sin x13)) x12) (- (* x7 x24) x3)))) (cos x17)) (/
x16 (- (* x24 x18) (- (sin (- (cos x13) (sin x13))) x3)))) (sin (- (max
(* (+ (sin (- (min (min (- (max x26 x29) (- x17 x28)) (* x17 x8)) x17) (*
x7 x24))) (max (sin x30) (max (+ x17 (min x13 x7)) (- (+ x17 x2) x26))))
(- x8 (sin x30))) (* (/ (sin (- (max x26 x29) x7)) x23) (- x22 (- x17
x28)))) (max x26 (- (max x26 x29) (min (max (+ (cos x8) x26) (max (+ x17
x2) (- (min x9 (min (* (min (sin x30) x20) x17) x16)) (- x17 x28)))) (max
x3 (* (- x17 x28) (* x24 x18))))))))))
```

Figure 7: The Best Individual.

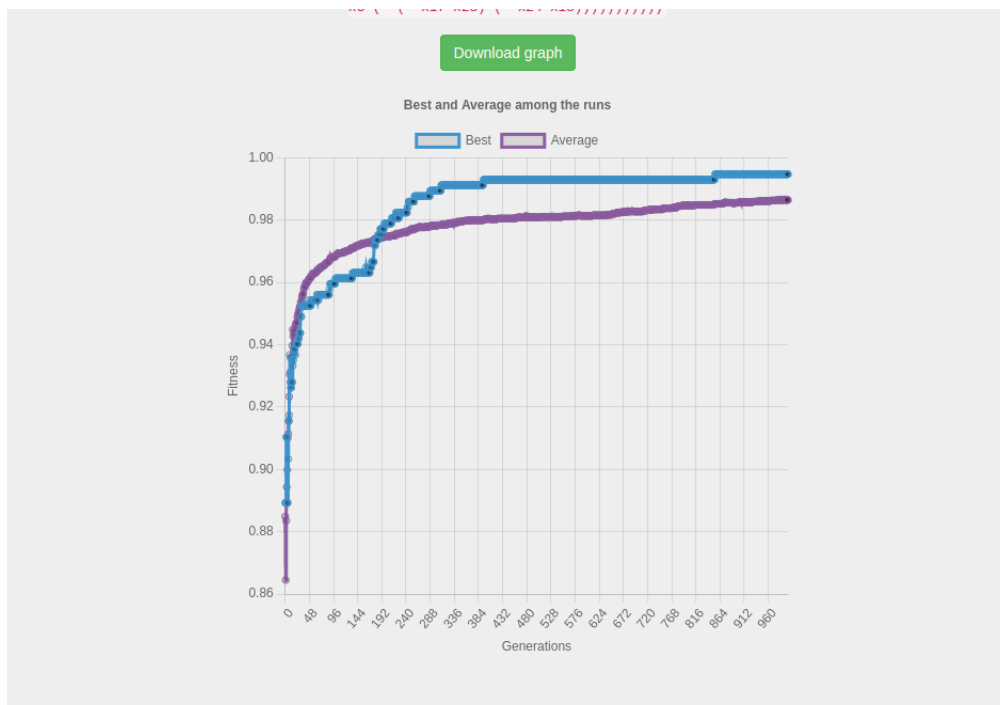


Figure 8: The Generation vs. Fitness Graph