Bachelor thesis

Differentiable Cartesian Genetic Programming as a Service

Mike Heddes

Amsterdam
June 21, 2019

Differentiable Cartesian Genetic Programming as a Service

Mike Heddes, 500735300

Amsterdam
June 21, 2019

European Space Agency
Advanced Concepts Team
Noordwijk, The Netherlands
Dr. Marcus Märtens

Amsterdam University of Applied Sciences
Engineering, Design and Innovation
Amsterdam, The Netherlands
Drs. Daria Meijers

Preface

In early 2015 I decided that I want a website on which I could exhibit my self produced music. The journey of making my own website started with Wordpress which offers some customization of their templates but I found myself reaching the limits its customization options quite early on. I was/am too stubborn to settle with the limitations that Wordpress has and started looking for a different solution. I started to think about how companies like Facebook make their website and I found out about Facebook's React JavaScript library. I read into React and JavaScript in general and became confident that this would be the way to tailor my own website to my taste.

Two years later when I got close to finishing my website[1] developed using React, I started wondering whether it would be possible to get a web programming (a.k.a. frontend developer) job with my acquired skill set. To my surprise I got a job interview at Zamro[2] which was the first time I could talk about my website with a professional. The senior developer that was doing my interview was impressed by what he saw and offered me a position. During my time at Zamro I learned a lot about writing good code and developing in teams. After three months in the company I got assigned the roll of maintainer of an internal project which meant that I needed to lead all the frontend teams to collaborate on one project. All of this was happening in parallel to studying Engineering, Design and Innovation (a.k.a. Mechanical Engineering) at the Amsterdam University of Applied Sciences.

I applied to the European Space Agency (ESA) for my graduation internship and got a response from Marcus, a research fellow in the Advanced Concepts Team (ACT), saying that he would like to interview me because of my interesting skill set. The internship assignment revolved around creating an easy to use application that provides value for engineers and scientists by producing the equation mapping between inputs and outputs. My knowledge of engineering, which I accumulated during my study, and my knowledge of coding proved to be a necessity to create this application.

I want to thank Marcus Märtens and Dario Izzo from the ACT for helping me during the internship. The ACT has been an experience unlike anything I had done before, I learned a lot from interacting with the team members and by simply being around them. I really enjoyed my time in the ACT and am excited for what the future will hold.

Mike Heddes, June 2019

[1] My personal website: https://mikeheddes.nl.
[2] Zamro webstore: https://zamro.nl.

Summary

Standard machine learning approaches like neural networks can take data and perform regression to extrapolate and interpolate behavior, the neural network itself is difficult to understand which makes it difficult to trust. Scientists in the Advanced Concepts Team (ACT) have developed a machine learning framework called differentiable Cartesian Genetic Programming (dCGP) (Izzo, Biscani, & Mereta, 2016), which provides explicit equations that can be much easier understood and studied by scientists and engineers. dCGP thus provides a form of explainable artificial intelligence, which can be applied to any supervised learning task.

dCGP was only available as a C++ or Python code library. This may prevent projects from using dCGP that could benefit from it because not every engineer has the prerequisite knowledge on C++ or Python. The goal of this project is to make it more convenient to use dCGP and to make tools that provide insight in the evolution and the resulting expression and equations. These two goals are grouped under the phrase "providing dCGP as a service". The main question to answer with this project is: *how can one best provide dCGP as a service?*

A mix of field, experimental and desk research has been done during this project. The field research comprises of interviews with clients, my supervisor and potential users. The desk research comprises of reading the documentation of relevant technologies and tools and reading papers on relevant topics. Dr. Märtens provided two documents related to dCGP and gave an introductory lecture about dCGP. In addition, experimental research was done to get a better understanding of how dCGP works and what it can do which also helped with getting a users perspective of the dCGP library.

First the problems with dCGP were concluded which are its unknown reliability, lack of insight and a high level of difficulty. Possible technologies and approaches to making the service were researched and laid out in a choice graph. The best choice was selected using weighted selection criteria in and selection matrix. The best approach was determined to be a client-side application which uses the React JavaScript library to make the UI and uses the to WebAssembly compiled C++ dCGP library to interact with dCGP on the client.

The service was developed using an iterative design method which helped reduce time spend on unimportant features and provided a tangible prototype early on in the development process. The implementation of the service is tested with automated unit tests that run before every change to the source code to make sure the code will still meet the requirements. The service is validated by the members of the ACT who would experiment and find bug after every incremental release.

Using dCGP is now effortless for every engineer, simply by navigating to the website of the service, uploading data and evolving. This opens up the possibilities for many projects to use dCGP to do analysis on data. Genetic programming scientists can now use the service to quickly test their cases and to get an intuition of how an evolution behaves over time.

To provide users with a better intuition on how the graph structure of dCGP works a view of the graph can be added to the application. The graph view can show which kernels are used and which nodes are connected. Each node could show its value so it is easier to reason about how the output is formed based on the intermediate values. In addition to the $ES-(\mu+\lambda)$ and gradient descent algorithms an algorithm combining both can achieve potentially better results in a shorter time. The hybrid algorithm should perform gradient descent before validating the mutated chromosomes.

Samenvatting

Standaard machine learning technieken, zoals neurale netwerken, kunnen gegevens opnemen en regressie uitvoeren om gedrag te extrapoleren en interpoleren, het neurale netwerk zelf is moeilijk te begrijpen waardoor het moeilijk te vertrouwen is. Wetenschappers in het Advanced Concepts Team (ACT) hebben een machine learning framework ontwikkeld, genaamd differentiable Cartesian Genetic Programming (dCGP) (Izzo et al., 2016), dat expliciete vergelijkingen biedt die veel gemakkelijker te begrijpen en te bestuderen zijn door wetenschappers en ingenieurs. dCGP biedt dus een vorm van verklaarbare kunstmatige intelligentie, die kan worden toegepast op elke supervised learning opdracht.

dCGP was alleen beschikbaar als een C++ of Python codebibliotheek. Dit kan voorkomen dat projecten dCGP gebruiken die er baat bij kunnen hebben, omdat niet elke ingenieur over de vereiste kennis van C++ of Python beschikt. Het doel van dit project is om het toegankelijker te maken om dCGP te gebruiken en om hulpmiddelen te maken die inzicht bieden in de evolutie en de resulterende expressie en vergelijkingen. Deze twee doelen zijn gegroepeerd onder de term “dCGP als een service aanbieden.” De belangrijkste vraag om met dit project te beantwoorden is: *hoe kan dCGP het beste als een service geboden worden?*

Tijdens dit project is een mix van field-, experimental- en deskresearch gedaan. Het fieldresearch bestaat uit interviews met klanten, mijn leidinggevende en potentiële gebruikers. Het deskresearch bestaat uit het lezen van documentatie van relevante technologieën en hulpmiddelen en het lezen van artikelen over relevante onderwerpen. Dr. Märtens gaf twee documenten met betrekking tot dCGP en gaf een inleidende lezing over dCGP. Daarnaast is experimenteel onderzoek gedaan om een beter beeld te krijgen van hoe dCGP werkt en wat het kan doen, wat ook hielp bij het verkrijgen van een gebruikersperspectief van de dCGP-bibliotheek.

Eerst werden de problemen met dCGP geconcludeerd, dit zijn de onbekende betrouwbaarheid, gebrek aan inzicht en een hoge moeilijkheidsgraad. Mogelijke technologieën en uitvoering voor het maken van de service zijn onderzocht en weergegeven in een keuzegrafiek. De beste keuze werd geselecteerd met behulp van gewogen selectiecriteria in en selectiematrix. De beste uitvoering was een client-side applicatie die de React JavaScript-bibliotheek gebruikt om de UI te maken en de met WebAssembly gecompileerde C++ dCGP-bibliotheek gebruikt om te communiceren met dCGP op de client.

De service is ontwikkeld met behulp van een iteratieve ontwerpmethode welke heeft geholpen om weinig tijd aan onbelangrijke functies te besteden. Volgens de ontwerpmethode is er vroeg in het proces een prototype gemaakt. De implementatie van de service is getest met geautomatiseerde unittests die vóór elke wijziging van de broncode worden uitgevoerd om te controleren of de code nog steeds aan de eisen voldoet. De service wordt gevalideerd door de leden van het ACT die na elke incrementele uitgave experimenteren en fouten zochten.

Het gebruik van dCGP is nu moeiteloos voor elke technicus, simpelweg door naar de website van de service te gaan, gegevens te uploaden en te evolueren. Dit opent de mogelijkheden voor veel projecten om dCGP te gebruiken voor analyse van gegevens. Genetische programmeerwetenschappers kunnen nu de service gebruiken om het voor hun toepassingen snel te testen en een intuïtie te krijgen van hoe een evolutie zich in de loop van de tijd gedraagt.

Om gebruikers een betere intuïtie te bieden over hoe de grafische structuur van dCGP werkt, kan een weergave van de grafiek aan de service worden toegevoegd. De grafiekweergave kan laten zien welke kernels worden gebruikt en welke knooppunten zijn verbonden. Elk knooppunt zou de waarde ervan kunnen tonen, zodat het gemakkelijker is om te redeneren over hoe de uitvoer wordt gevormd op basis van de tussenliggende waarden. In aanvulling op de $ES-(\mu+\lambda)$ en gradiëntdaling algoritmen kan een algoritme dat beide combineert potentieel betere resultaten bereiken in een kortere tijd. Het hybride algoritme zou gradiëntdaling moeten uitvoeren voordat de gemuteerde chromosomen worden gevalideerd.

Table of Contents

Preface ii

Summary iii

Samenvatting iv

Terminology viii

1 Introduction 1

2 Problem analysis 2

2.1 Problems . . . 2
2.2 Orientation resources . . . 2
2.3 Research questions . . . 2
2.4 Relevance . . . 3
2.5 Stakeholders . . . 3
2.6 Scope . . . 4
2.7 Client feature requests . . . 4

3 Solution analysis 5

3.1 Starting point . . . 5
3.2 Differentiable Cartesian Genetic Programming . . . 5
3.3 Target audience . . . 6
3.4 Controls and functionality . . . 6
3.5 Application development options . . . 7
3.6 Serving a web application . . . 7
3.7 dCGP on the server . . . 8
3.8 dCGP on the client . . . 8
3.9 Markup generation . . . 9
3.10 Application inputs and outputs . . . 10
3.11 Requirements . . . 10

4 Technology selection 12

4.1 Application approach . . . 12
4.2 Frontend framework . . . 12

5 Detailing **14**

5.1 Tools . . . 14

5.2 Iterative design . . . 14

5.3 Cases . . . 15

5.4 GitHub repositories . . . 16

5.5 Tests and validation . . . 16

5.6 Documentation . . . 17

6 Realizing **18**

6.1 Distribution . . . 18

6.2 Marketing . . . 18

6.3 Maintenance . . . 18

7 Conclusion **19**

8 Recommendations **19**

References **20**

Appendices **21**

A Differential Cartesian Genetic Programming **22**

A.1 What dCGP does . . . 22

A.2 How dCGP works . . . 22

A.3 What dCGP can be used for . . . 22

B UI design **24**

C WebAssembly bindings **27**

C.1 Dependency compilation . . . 27

C.2 Provided API . . . 27

C.3 Memory management . . . 28

D Web Worker system **29**

D.1 Proxy . . . 29

D.2 Observables . . . 29

D.3 Consistent framerate . . . 30

E Bugs **31**

E.1 Protected division . . . 31

E.2 Evolution loop . . . 31

F Software development platform interest **32**

G Unit tests **33**

H Service validation **34**

I Interviews **37**

I.1 Introduction to dCGP as a service 37

I.2 dCGP service, users and functionality 37

I.3 dCGP web service feedback 37

List of Figures

3.1 dCGP service function diagram 6
3.2 Possible combinations to make the service 7
4.1 The combination that best fitted the project's selection criteria. 13
5.1 dCGP service final UI design. 15
5.2 dcgp.js expression documentation page. 17
A.1 Image encoding comparison 23
B.1 dCGP service UI design iteration one. 24
B.2 dCGP service UI design iteration two. 25
B.3 dCGP service UI design iteration three. 26
F.1 Software development platform interest 32
H.1 dCGP service validation without the cosine and sine kernels. 34
H.2 dCGP service validation with the cosine and sine kernels. 35
H.3 dCGP service validation without the sine kernel and with a constant. 36

List of Tables

0.1 Acronyms, abbreviations and terms overview viii
2.1 Research questions and type. 3
2.2 Client feature requests. 4
3.1 Service requirements 11
3.2 Selection criteria weights. 11
4.1 Comparison of web application approaches 12
4.2 Comparison of frontend frameworks. 13

Terminology

Table 0.1: Overview of the acronyms, abbreviations and terms used in this document.

Term	Full name or description
ACT	Advanced Concepts Team
ANN	Artificial Neural Network
API	Application Programming Interface
Arity	The number of arguments that the kernel takes
AuDi	Automated Differentiation
C++	General-purpose low-level programming language
CGP	Cartesian Genetic Programming
CI	Continuous Integration
Client	Client of the project or the computer of the user that is visiting the service.
CRA	Create React App
CSS	Cascading Style Sheets
dCGP	differentiable Cartesian Genetic Programming
DOM	Document Object Model
Ephemeral constants	Constants that are guessed by the user in the hope that the chromosome will evolve the constant to get the correct value.
ESA	European Space Agency
GP	Genetic Programming
Gdual	Generalized dual number
HTML	Hypertext Markup Language
JS	JavaScript
Python	Interpreted, high-level, general-purpose programming language
PWA	Progressive Web App
Supervised learning	Task of learning a function that maps an input to an output based on example input-output pairs.
UI	User Interface

1 Introduction

Enabling access to space is one of the goals[3] of the European Space Agency (ESA) and requires a lot of engineering effort and tests. Each satellite is examined by simulating the extreme conditions that appear during launch and within orbit. The data collected during tests help to understand how materials will react to extreme conditions, how energy or heat gets dissipated, how much thrust is needed to align a satellite and much more. While standard machine learning approaches like neural networks can take this data and perform regression to extrapolate and interpolate the behavior, the neural network itself is difficult to understand which makes it difficult to trust. Scientists in the Advanced Concepts Team (ACT) have developed a machine learning framework called differentiable Cartesian Genetic Programming[4] (dCGP) (Izzo et al., 2016), which provides explicit equations that can be much easier understood and studied by scientists and engineers. dCGP thus provides a form of explainable artificial intelligence, which can be applied to any supervised learning task.

The goal of this six-month project is to make it more convenient to use dCGP and to make tools that provide insight in the evolution[5] and the resulting expression and equations. These two goals are grouped under the phrase "providing dCGP as a service". The underlying idea is that dCGP will move one step further from an academic experiment towards a product (software tool) helpful for engineering.

The main question to answer during this project is: **how can one best provide dCGP as a service?** To answer the main question the following sub questions need to be answered:

1. What does dCGP do?
2. How does dCGP work?
3. What can dCGP be used for?
4. Who are interested in dCGP as a service?
5. Which controls should be available on the service?
6. How can the service integrate the dCGP framework?
7. Which technologies can be used to make the service?
8. What requirements must the service meet?

To answer the previously stated sub questions a mix of field, experimental and desk research is used. The field research comprises of interviews with clients, my supervisor and potential users. Feedback on the service by early adopters is also seen as field research. Experimental research is done to get a better understanding of how dCGP works and what it can do. The desk research comprises of reading the documentation of relevant technologies and tools and reading papers on relevant topics such as genetic programming and the dCGP paper.

The remaining part of this thesis covers the problem analysis in chapter 2 which describes what the problem is, who the stakeholders are and which features the clients want the service to have. Chapter 3 states the research that has been conducted about dCGP, the target audience and the technologies that can be used to make the service. Chapter 4 describes and substantiates the choices that were made on the implementation of the service. Chapter 5 describes the development process of making the service. Chapter 6 describes the distribution, marketing and maintenance of the service. Chapter 7 states the conclusion of the thesis. Chapter 8 provides recommendations to improve the service. Appendix A describes how the dCGP framework works and what it can be used for. Appendix B contains the UI design iterations. Appendix C covers an implementation case about compiling the dCGP framework to WebAssembly. Lastly, appendix D covers an implementation case about the use of Web Workers in the service.

[3] What is ESA web page: https://www.esa.int/About_Us/Welcome_to_ESA/What_is_ESA.

[4] dCGP is a subset of Genetic Programming (GP) which is a field of computer science that uses biology metaphors to frame optimization tasks.

[5] In GP evolution is an iterative training process that can change the encoded expression to optimise for a given evaluation. This is similar to evolution in biology where evolution can change the DNA of a population to optimise for survival.

2 Problem analysis

This chapter describes three problems that are present in the state of dCGP prior to this project. In addition the relevance, stakeholders and boundary conditions of the project followed by the goals of the client are stated.

2.1 Problems

In an interview with Dr. Märtens (Heddes, 2019c) two problems surfaced from the ACT's perspective. The first problem results from the high standards ESA sets for their tools. Within ESA all tools used for mission critical applications need to be reliable and precise because small errors during the development of a mission can have major implications for the success of the mission. dCGP is a relatively new framework with few people actively testing it therefore the reliability has not been proven and thus can dCGP not be used for mission critical applications yet.

The encoding of the expression used in Cartesian Genetic Programming (CGP) is made to be computer memory efficient and easy to manipulate by evolution algorithms. The encoding was not made to be readable for humans. Inspecting the expression is therefore a confusing and time-consuming task. There are, as of writing this thesis, no tools available that can provide scientists a way of easily inspecting what is encoded in the expression.

dCGP has a steep learning curve because it tries to solve an inherently difficult problem and it requires prerequisite knowledge about programming in C++ or Python[6] which not every engineer will have. The steep learning curve may prevent projects from using dCGP that could benefit from the functionalities that dCGP offers.

In summary, the three problems dCGP had are unknown reliability, lack of insight and a high level of difficulty. This project mainly focused on the third problem and added features that will help resolve problem two as well.

2.2 Orientation resources

In the first week of the project Dr. Märtens provided two documents related to dCGP. The first document, *Differentiable Genetic Programming* (Izzo et al., 2016), is the original paper introducing dCGP. The second document, called *Evolving Artificial Neural Networks using Cartesian Genetic Programming* (Turner, 2015), is a PhD thesis describing many aspects of CGP and its relation to ANN's. Along with the documents Dr. Märtens gave an introductory lecture about dCGP. In addition, experimental research was conducted to get a better understanding of how dCGP works and what it can do. This meant coming up with a superficial example to use dCGP on. Doing the experimentation provided a users perspective of the dCGP library. Lastly, the original paper introducing CGP, simply called *Cartesian Genetic Programming* (Miller & Thomson, 2000), was used to understand how the underlying CGP works. The result of the experimentation together with a comprehensive explanation of dCGP is stated in appendix A.

2.3 Research questions

The goal of this project was to make it more convenient to use dCGP and to make tools that provide insight in the evolution and the resulting expression and equations. These two goals were grouped under the phrase "providing dCGP as a service". The underlying idea was to move dCGP one step further from an academic experiment towards a product (software tool) helpful for engineering.

The main question to answer during this project was: **how can one best provide dCGP as a service?** In order to answer the main question the sub questions stated in table 2.1 needed to be answered. The next list substantiates the decisions of the specified research types in table 2.1.

1. By reading papers on dCGP and GP an insight into what dCGP does was formed.

[6] The dCGP framework is only available as a C++ or Python library.

2. By reading papers on dCGP and GP and by doing an experiment, an insight into how dCGP works was formed.
3. By reading papers on dCGP and GP and by having conversations with machine learning scientists an insight into what dCGP can be used for was formed.
4. By doing an interview with Dr. Märtens an insight into their target audience was formed.
5. By doing an interview with Dr. Märtens and discussions with Dr. Izzo an insight into which controls the service should have was formed.
6. By reading into established and new ways to integrate software in a web application an insight into the possible technologies to integrate the dCGP framework was formed.
7. By searching for and reading into promising technologies (Green & Seshadri, 2013) and by watching conference talks by web industry experts (Occhino & Walke, 2013; You, 2017) an insight into which technologies can be used to make the service was formed.
8. By doing an interview with Dr. Märtens and reading papers on dCGP the requirements of the service were determined.

Three interviews that were conducted during this internship are described in appendix I.

Table 2.1: Research questions and type.

#	Research question	Research type
1.	What does dCGP do?	Desk
2.	How does dCGP work?	Desk and experiment
3.	What can dCGP be used for?	Desk and field
4.	Who are interested in dCGP as a service?	Field
5.	Which controls should be available on the service?	Field
6.	How can the service integrate the dCGP framework?	Desk
7.	Which technologies can be used to make the service?	Desk
8.	What requirements must the service meet?	Desk and field

2.4 Relevance

Making dCGP more approachable and easier to use opens up the possibilities for various industries to use dCGP to find mathematical relations between data. In the mechanical industry dCGP can be used to find relations between any numerical data to optimise existing products or to find new opportunities to explore which can catalyze the development of new technologies.

Each satellite at ESA is examined by simulating the extreme conditions that appear during launch and within orbit. The data collected during tests help to understand how materials will react to extreme conditions, how energy or heat gets dissipated, how much thrust is needed to align a satellite and much more. Making dCGP more approachable means that it can be used more frequently to understand the data that is collected during tests.

dCGP is also used in the industry by the DowDuPont company. DowDuPont manufactures, among other things, plastics and chemicals. The research and development department of DowDuPont uses dCGP to find the relation between material properties of their polymers. Testing exotic polymers is very expensive which makes it essential to be able to extract the relation from as few measurements as possible. By finding the relation between material properties DowDuPont can manufacture polymers optimized for a specific task.

2.5 Stakeholders

The following list specifies the stakeholders of the project with a description stating their relation to the project.

- Dr. D. Izzo, the ACT scientific coordinator and the creator and maintainer of the dCGP framework written in C++ and Python.
- Dr. M. Märtens, research fellow in the ACT in the field of artificial intelligence. Dr. Märtens is the supervisor of this project and will become the maintainer.
- Users of the service, which can be the general public because ESA decided that the project will be open source and usable for free.

2.6 Scope

The following points fall within the scope of this project:

- Research ways of implementing dCGP in a service.
- Designing an interface for the service.
- Developing the data infrastructure for the service.
- Delivering a working service.

The following points fall outside the scope of this project:

- Maintaining or extending the dCGP framework or its dependencies.
- Maintaining or extending technologies used to build the service.
- Creating an environment on which the service can be distributed.

2.7 Client feature requests

Table 2.2 contains a list of features that Dr. Märtens and Dr. Izzo, the clients of this project, would like to see implemented in the service. The second column specifies the type of the feature, this can either be a requirement, success criteria or *nice to have*. *Nice to have* means that the clients would like the service to have this functionality but it is not required to be added within this project.

Table 2.2: Client feature requests.

#	Feature	Type
1.	Users can access most of the functionalities that dCGP offers.	Success criteria
2.	The service can be used with minimal actions required by the user.	Success criteria
3.	The service can be used with minimal external dependencies.	Success criteria
4.	The service provides a way to view the encoded equation.	Requirement
5.	The service provides a way to view the evolved network topologies.	*Nice to have*
6.	The service provides a way to inspect a chromosome.	*Nice to have*

3 Solution analysis

This chapter contains the research conducted during this project meant to answer the sub questions stated in section 2.3.

3.1 Starting point

This section describes the decisions that were made early on in the project. These decisions are seen as a given in this report which means no further reasoning or substantiation will be given for these choices. These decisions were made in consultation with supervisor Dr. Märtens.

3.1.1 User interface

The service will be a user interface (UI) rather than a code library. This decision was made based on the following insights.

1. The original dCGP project was implemented in C++ with bindings for Python. Creating another binding library will add little value since most scientific computing is done using Python for which bindings already exist.
2. A UI is more approachable for the general public because it is familiar and does not require reading into the project to get started.

3.1.2 Web application

The service will be a web application rather than a native application. This decision was made based on the following insights.

1. Native applications can only be made for one platform because all platform manufacturers (Apple, Google, Windows) use different languages and software development kits (SDK) to develop an application. There is no standardization because every platform has its own SDKs which means that one would have to relearn everything when switching to a different platform. The W3C standards make sure that all browsers follow the same application programming interface (API) implementation in JavaScript.
2. Native applications are written in lower level languages than JavaScript like Objective-C on iOS and macOS, Java on Android and C++ on Windows. Using lower level languages makes the code in general harder to understand and reason about.
3. Native applications in general are faster because they are written in a lower level language. However there is a new web technology called WebAssembly which makes it possible to execute low level code on the web at near native speed. Another way could be to run the computation heavy tasks on the server where the fastest native implementation can be used. Ways of running the dCGP code are further discussed in sections 3.7 and 3.8.
4. Web applications have a higher chance of getting contributors because the entry level of making a JavaScript application is lower than making a native application. In addition, according to Stack Overflow[7] there are more JavaScript developers then there are for any platform specific language.

3.2 Differentiable Cartesian Genetic Programming

This section answers three research questions, *what does dCGP do*, *how does dCGP work* and *what can dCGP be used for*. At its core Cartesian Genetic Programming (CGP) provides a way to encode a representation of a computational graph as a chromosome (see Turner, 2015, para. 3.2). dCGP adds the functionality of differentiating the encoded equations which makes it possible to learn constants. The chromosome is a list of numbers that contains the identifiers of the inputs and the kernel identifier for each node. An evolution algorithm can change the numbers in the chromosome to alter the encoded equation, this is called symbolic regression. Often the goal is to find an equation that closely resembles

[7]Stack Overflow developer survey: https://insights.stackoverflow.com/survey/2018.

the mapping between numerical input and output data. Appendix A covers what dCGP is, how it works and what it can be used for in greater detail.

3.3 Target audience

This section answers the research question *who are interested in dCGP as a service*. From an interview with Dr. Märtens (Heddes, 2019a) it became clear that the target audience is interested in dCGP and might have little to very advanced knowledge about dCGP. Dr. Märtens pointed out that the service should at least be usable for advanced users so the service can be used internally. In addition, he envisions that the service will be used to showcase dCGP to stakeholders and directors to create excitement and curiosity. Lastly, he mentioned that the service could be used as an introduction and a playground for people that are new to the dCGP framework to get a feeling of what dCGP does and is capable of.

The main target audience for this project are scientists and engineers that frequently work with or make measurements during their work. This for example applies to material scientists, bio engineers and many more disciplines. For many of these disciplines finding the equational relation between measured properties is more interesting than the measurements themselves. So far it has however been quite hard and time consuming to find the equational relation between properties from measurements. Providing an accessible and easy to use service on which dCGP can effortlessly be used allows these scientists and engineers to do their work faster and will convince more engineers and scientists to use dCGP.

It seemed reasonable to assume, based on the nature of the original dCGP project, that the target audience has at least basic knowledge of computer science and mathematics and is comfortable navigating around in modern web applications. The education level of the target audience is set to be at least a bachelor's degree. The target audience consists mostly of engineers that want a quick and easy to use application to find the equational relation between inputs and outputs.

3.4 Controls and functionality

This section answers the research question *which controls should be available on the service*. Figure 3.1 is a function diagram that states the controls and functionalities that arose from an interview with Dr. Märtens (Heddes, 2019a). The top of the function diagram states the main function of the service which is *providing dCGP as a service*. This main function is then divided in subfunctions which are again divided in subfunctions to create a breakdown of all the functionalities of the service. Users of the service must be able to run an evolution algorithm, change the parameters of the evolution and the expression and, upload custom data.

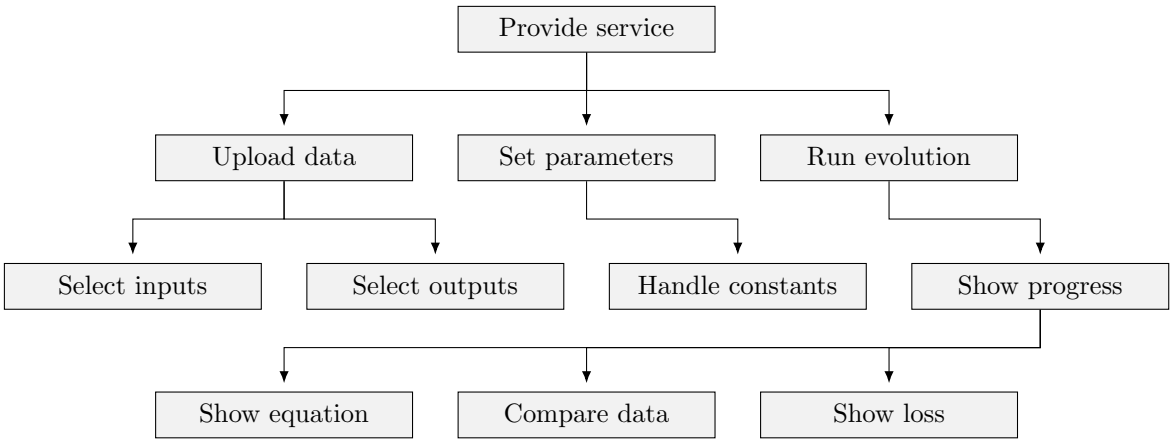

Figure 3.1: dCGP service function diagram

3.5 Application development options

This section answers two research questions, *how can the service integrate the dCGP framework* and *which technologies can be used to make the service*. The sections 3.6 till 3.9 describe different options for developing the service. These options led to the in figure 3.2 depicted possible combinations to make the service. The first row of the figure specifies the frameworks that can be used to make the web application, these are the Django Python framework, React JavaScript framework, Vue JavaScript framework and Angular JavaScript framework. The last row specifies the ways dCGP can be integrated in the application, these are using the Flask Python library, FastAPI Python library or by compiling dCGP to WebAssembly. The second row specifies different approaches to combine dCGP with a web application.

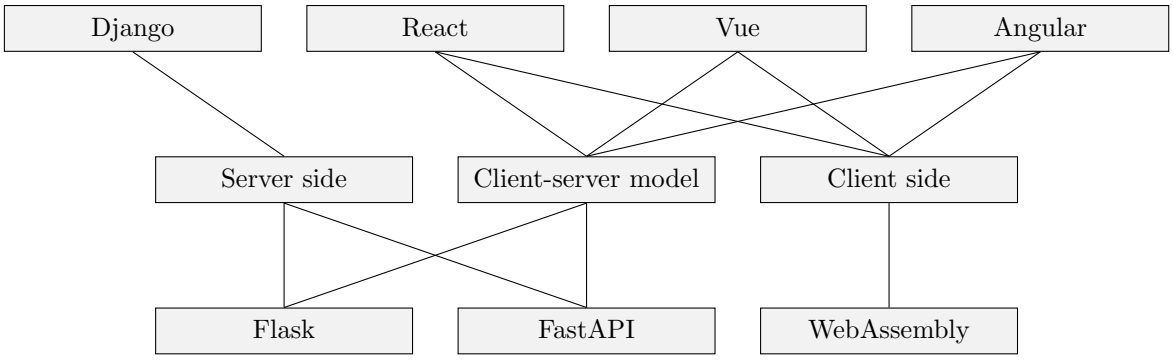

Figure 3.2: Possible combinations to make the service

3.6 Serving a web application

This section describes the different commonly used approaches for making a web application.

3.6.1 Server-side

With a server-side application the server handles all the logic of the application and the generation of the HTML page which is then send to the client to be displayed. The server application can be written in any language which makes it possible to use the existing C++ or Python dCGP implementation. Because all the logic of the application is on the server every user interaction will request a new page from the server which adds a significant networking delay on user interactions. This can break the illusion that the user is using a native application.

Hosting a server-side application will require continuous maintenance to update the software, certificates, user accounts and to fix security vulnerabilities. Some of the mentioned maintenance points can be taken care of by renting a server. However not all security vulnerabilities can be eliminated by renting a server, some vulnerabilities like cross-site-scripting and bugs in the application are possibly introduced by the code written for the application.

3.6.2 Client-side

With a client-side application the server sends all the logic of the application to the client including the logic to generate the HTML of the page. This has the benefit of being able to handle user interactions immediately, which gives the application a native like experience. Client-side code is restricted to JavaScript and WebAssembly[8] which means that the current dCGP implementations can not be used directly and will need to be ported to either WebAssembly or JavaScript.

[8]WebAssembly can not access the document object model (DOM) of the page. This means that only JavaScript can be used to make changes to the page. WebAssembly can be used to do general calculations.

Not having a central server application that keeps track of the users makes it hard to send information between users. However with a client-side application the server only needs to serve static content which makes renting a server free in the case of some server providers[9] and eliminates maintenance. Having the logic of the application be executed on the client also prevents security issues. For example, when an infinite loop occurs the client is in control and will close or reload the page. This is still unfortunate for the one user but does not influence the experience of other users which is the case if it would happen on the server.

3.6.3 Client-server model

With a client-server model there are two applications running, one on the server and one on the client. The server-side application is responsible for the core logic of the application, which is dCGP in this case. The client-side application is responsible for displaying the content provided by the server via an API and for handling user interactions. A client-server model has the benefits of separating the concerns and being able to make the server application in any language so the current dCGP implementation can be used while still being able to immediately respond to user interactions.

Using a client-server model comes with the same server related drawbacks as a server-side application. For corporate sized web applications the client-server model is considered the standard method of making a web application because it scales well and provides a way of keeping sensitive company logic on the server.

3.7 dCGP on the server

In this section two options for hosting an API web server with Python are described. Only Python is considered because between C++ and Python, the dCGP implementations, Python has a simpler syntax and developing an application with Python will take considerably less time than in C++ although the C++ implementation will execute faster.

3.7.1 Flask

Flask is the most known and widely used Python web framework. It is able to perform all the common tasks a web API needs to do but lacks a build-in solution for WebSockets. Flask has a simple API and a large community with many resources to learn from online. Flask's API can easily be connected with the dCGP library to make the resolvers for the API endpoints. Because of the large user base and intensive testing Flask has proven to be reliable with many use cases.

3.7.2 FastAPI

FastAPI promotes itself as being a faster alternative to Flask. FastAPI is based on the Starlette library which is based on the uvicorn library. A third-party benchmark[10] shows that uvicorn is the fastest Python framework followed by Starlette and FastAPI with 86.3% and 72.5% of the performance respectively. The FastAPI provides an easy to use high level API with integrated data validation and serialization, it also provides automated API documentation. It however has a much smaller community than Flask which results in fewer resources and a less extensively tested codebase.

3.8 dCGP on the client

This section describes the different ways the original dCGP implementation can be used on the client in the case of a client-side application. The dCGP library can be rewritten in JavaScript, the C++ source code can be compiled to JavaScript or, the C++ source code can be compiled to WebAssembly. The next section will look at the WebAssembly approach because it provides the fastest execution and makes it possible to reuse the existing C++ dCGP source code.

[9] GitHub provides free hosting for static sites.

[10] Python web framework benchmark: https://www.techempower.com/benchmarks/#section=test&runid=7464e520-0dc2-473d-bd34-dbdfd7e85911&hw=ph&test=query&l=zijzen-7.

3.8.1 WebAssembly

This section describes WebAssembly and the different methods that are available to port the C++ dCGP library to WebAssembly.

WebAssembly was introduced in *Bringing the Web up to Speed with WebAssembly* (Haas et al., 2017) which was a collaboration between all major web browsers (Chrome, Firefox, Safari and Edge) to "address the problem of safe, fast, portable low-level code on the Web". WebAssembly is a newly developed assembly-like low-level byte-code language. The idea of WebAssembly originated from the by Mozilla created JavaScript subset called asm.js which used specific JavaScript syntax that the browser's JavaScript compiler could efficiently and effectively optimise to get near native speed.

The first option to port the C++ dCGP library to WebAssembly is to compile the original C++ source code to WebAssembly using Emscripten. "Emscripten is a toolchain for compiling to asm.js and WebAssembly, built using LLVM, that lets you run C and C++ on the web at near-native speed without plugins." as described by the Emscripten documentation[11]. Emscripten was first introduced in *Emscripten: An LLVM-to-JavaScript Compiler* (Zakai, 2011) and was developed alongside the creation of asm.js to provide a way to generate asm.js code from C or C++. Since WebAssembly was created Emscripten has updated to also allow WebAssembly as a compilation target. Emscripten provides many helper functions and tools for porting existing C or C++ projects to WebAssembly.

An alternative to Emscripten is called *Minimal WebAssembly* which describes itself as "a minimal toolkit and runtime ... to produce and run WebAssembly modules". This toolkit has the advantage of being easier to get started with because the API surface is smaller than that of Emscripten. Having a smaller API comes with the drawback that a lot of things that Emscripten already provides need to be coded by hand.

3.9 Markup generation

This section describes four different frameworks that are available for developing interactive web applications. The four frameworks covered are Django, React, Vue and Angular.

3.9.1 Django

Django is a Python web framework that provides an all-in-one solution for web applications. It handles everything from user authentication, databases to markup generation. Django is an easy to use framework that can help speedup the process of developing a web application. Django runs fully on the server where it generates the HTML and sends that to the client. Because of the many features Django provides the framework is bloated and has a large API surface. Django is an opinionated framework which leaves the developers working with Django little freedom in designing the application architecture. Django is thoroughly tested by its many contributors and users.

3.9.2 React

React is a JavaScript frontend UI framework created by Facebook. React provides a declarative way of writing reusable components (Occhino & Walke, 2013). It is currently the most popular frontend framework, used in production by Facebook and Instagram. The large community behind React provide many resources to learn from and tools to build applications faster. React is unopinionated which means it can be combined with any other library. Combining React with a data handling library like Redux or MobX is a common setup. React provides tools to test the components which is necessary to make a framework a scalable solution. React is often written in jsx which provides HTML-like syntax in JavaScript. This has the disadvantage that new developers need to learn the subtle differences between jsx and HTML.

[11] Emscripten documentation: https://emscripten.org/.

3.9.3 Vue

Like React, Vue is also a JavaScript frontend UI framework but instead of writing jsx it works with HTML templates that it binds to JavaScript objects (You, 2017). Vue is reactive which means that changes in the data of the object will result in changes in the view. The size of the Vue library is considerably smaller than that of React. A disadvantage of Vue is its small community which results in fewer helper packages available to speed up development. Vue lacks large corporate backing which might help explain its smaller community.

3.9.4 Angular

Other than Vue and React, Angular is an opinionated JavaScript framework which provides everything necessary to make a view and to handle complex data (Green & Seshadri, 2013). Because it is opinionated it lacks the flexibility that Vue and React have. Angular has a steep learning curve because of its large API surface and complex data binding. The techniques it uses became outdated since its launch in 2012. Angular is backed and developed by Google which makes it more likely that it will stay around for a longer time than Vue.

3.10 Application inputs and outputs

Inputs to the service consist mainly of user interactions, this can be via the mouse, keyboard or touch. Other inputs are the custom data a user can provide and the URL the user visits. The environment of the user can also be seen as an input to the application because it determines which features are available for the service to use. A dCGP expression takes eight arguments which are the number of inputs and outputs of the expression, the number of rows and columns of the graph, the number of levels back a connection can maximum be, the arity of each kernel, a list of the kernels and a seed.

The UI is the main output of the application, it consists of an equation view, the loss of the current expression with the labels, the amount of steps evolved and a plot with labels and predictions. Another output of the application is the equation as a string which can be copied and used for further research.

3.11 Requirements

This section answers the research question *what requirements must the service meet*. The requirements have been put together based on the previously described research and the goals of the client. The requirements are stated in table 3.1.

All requirements, apart from 5 and 7, were verified using automated unit tests that executed on every iteration to determine whether the code base meets the requirements. Requirement 5 was verified by visiting the web page and seeing if the functionalities were working. Requirement 7 was verified by opening the browser's debugger and reading the time between interaction and response.

3.11.1 Selection criteria

In addition to the requirements of the service the following selection criteria have been put together to act as a means of making substantiated decisions about which technologies to develop the service with.

1. Service is usable in minimal amount of steps.
2. Response time of the application is fast.
3. Reference to the service is easy to share with others.
4. Evolution runs fast.
5. Time till page load is short.
6. Maintainability of the code is good.
7. Used technology has many developers.

Table 3.1: Service requirements

#	Requirement	Source
1.	The application must be able to evolve an expression.	Miller and Thomson, 2000
2.	The application must provide the mu plus lambda evolution algorithm.	Turner, 2015, par. 3.4
3.	The application must provide the gradient descent algorithm.	Turner, 2015, par. 2.6.9
4.	The application must be able to show the generated equation.	Heddes, 2019a
5.	The application must work without dependencies to be installed by the users.	*Self imposed*
6.	The application must inform the user if any user input is invalid.	*Self imposed*
7.	The application must respond to user interactions in under 500 milliseconds.	*Self imposed*
8.	Users must be able to change which kernels are active.	Heddes, 2019a
9.	Users must be able to change the size of the expression.	Heddes, 2019a
10.	Users must be able to add ephemeral constants.	Izzo, Biscani, and Mereta, 2016
11.	Users must be able to remove ephemeral constants.	Izzo, Biscani, and Mereta, 2016
12.	Users must be able to change the value of the ephemeral constants.	Izzo, Biscani, and Mereta, 2016
13.	Users must be able to provide its own data for evolution.	Heddes, 2019a
14.	Users must be able to evolve on data without needing to upload data.	*Self imposed*

3.11.2 Selection criteria weights

Not all the selection criteria have the same importance for the service. In oder to reflect the importance differences in the decision making based on the selection criteria a weight has been assigned to every selection criteria. The weights range from 1 to 5 with the latter being the best score. Table 3.2 states the weight for every selection criteria followed by a description of why that weight has been assigned.

Table 3.2: Selection criteria weights.

#	Weight	Description
1.	5	This is essential to create an easy to use service.
2.	2	A slow responding application results in bad user experience but fast responses are not essential.
3.	4	This is essential to spread the service fast and to do effective marketing.
4.	3	It is important that the service runs as fast as possible but it is not essential to the application.
5.	1	The page load is a onetime payment by the user and is therefore less important then a fast application.
6.	2	Since the application will likely be maintained by less experienced developers it is important that the application is easy to maintain.
7.	1	Using a technology with many developers will help to find maintainers in the future but is not an important distinction.

4 Technology selection

This chapter describes and substantiates the decisions that were made related to the approach and framework used for the service.

4.1 Application approach

The first selection matrix determined the kind of application that was made. This could either be a server-side, client-side or client-server model application as described in section 3.6.

Table 4.1: Comparison of the following web application approaches, a server-side (S), client-side (C) or client-server model (CS) application.

Selection criteria	weight	S	C	CS
1. Service is usable in minimal amount of steps.	5	10	10	10
2. Response time of the application is fast.	2	1	8	6
3. Reference to the service is easy to share.	4	10	10	10
4. Evolution runs fast.	3	7	10	7
5. Time till page load is short.	1	10	4	7
6. Maintainability of the code is good.	2	5	8	2
7. Used technology has many developers.	1	2	10	10
Score		135	**166**	144

The selection matrix in table 4.1 shows that making the service as a client-side application would result in the best match with the selection criteria of the service. The score for each approach on every selection criteria is based on its relative strengths and weaknesses against the other approaches and against the theoretical best possible implementation. For instance, a server-side application has a very long time till interaction because it needs to make a network request on every interaction. A client-server model application improves this significantly by being able to respond to an interaction immediately while waiting for the results to return from the server. A client-side application improves this even further by sending all the related code to the client so no additional network requests need to be made to handle an user interaction. A client-side application does not score 10 points because sending all the related code to the client makes the first page load take longer but subsequent interaction can be handled immediately.

After choosing the client-side application approach there are still options as to which framework to use to make the application which will be selected in the next section.

4.2 Frontend framework

The second selection matrix determined the frontend framework that was used to make the application. This could either be React, Vue or Angular as described in section 3.9. Django is not compatible with a client-side application and is therefore not an option in this selection.

The selection matrix in table 4.2 shows that making the application using Vue or React would both be an equally good choice. In the end React has been chosen for the service because I am already familiar with it, which saves a lot of time spent on learning a new library and ecosystem. The score for each framework on every selection criteria is based on its relative strengths and weaknesses against the other frameworks and against the theoretical best possible implementation. For instance, Angular has the longest page load time because it is the biggest framework which needs to be send to the client. React is much better because the size of the framework is much smaller. Vue is an even smaller framework which reduces time till page load even more. Vue does not score 10 points because there are still bytes that need to be send to the client, a native application has the code for the application already on the client which makes the time till page load practically zero.

Figure 4.1 shows which technologies the resulting application will have.

Table 4.2: Comparison of frontend frameworks.

Selection criteria	weight	React	Vue	Angular
1. Service is usable in minimal amount of steps.	5	10	10	10
2. Response time of the application is fast.	2	8	10	4
3. Reference to the service is easy to share.	4	10	10	10
4. Evolution runs fast.	3	8	10	6
5. Time till page load is short.	1	5	7	1
6. Maintainability of the code is good.	2	7	5	2
7. Used technology has many developers.	1	10	2	6
Score		**159**	**159**	127

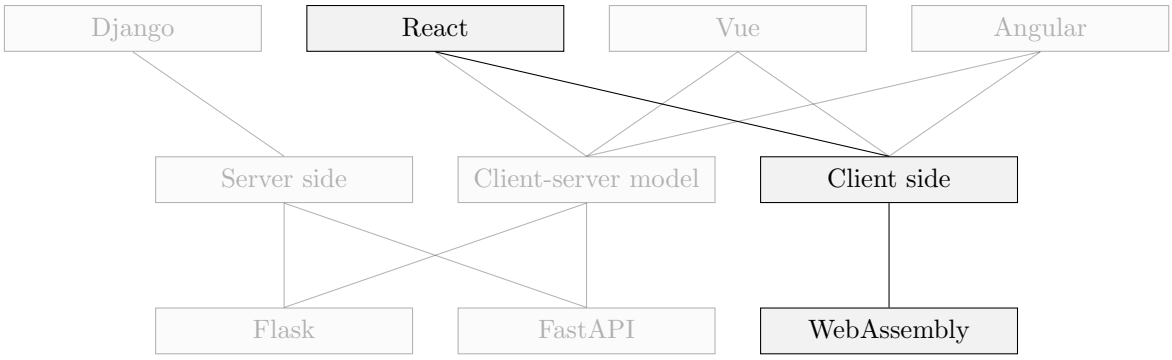

Figure 4.1: The combination that best fitted the project's selection criteria.

5 Detailing

This chapter describes the implementation process of the service, methods and tools used to make the application and three cases that describe interesting and challenging aspects of the application.

5.1 Tools

During development of the service the following tools were used.

- *Adobe XD* is used to create wireframes for the design of the application. Adobe XD provides an easy to use UI to draw shapes, place text and images and, assign colors. Creating designs is efficient which allows exploring more variations of a design.
- *CircleCI* is a continuous integration (CI) provider that is used to automate the repetitive tasks that come with validating, testing and releasing code. The CI system is triggered on every change on GitHub, for example, a pull request was made or the master branch was updated. In general, the tests and validation are run on pull requests to determine whether the code is valid, a new version of the code is released when the master branch updates.
- *Create React App* (CRA) is used to initialize a React project with all the boilerplate necessary to develop a React web application already in place. CRA is maintained by Facebook, it allows updating to the latest version after initializing a project[12] which removes the overhead of staying up-to-date with all the boilerplate libraries.
- *Git* is used to have version control of the code. At its core Git provides a way of attaching a unique identifier to a snapshot of the projects files, this is called a commit. Multiple commits can form a history. The snapshots are used to allow collaboration on coding projects by handling the merging of files based on their shared and separate history.
- *GitHub* is used as a source of truth from which code gets cloned to either develop or release. GitHub is a website that provides a UI for most of the Git commands. It also provides issue tracking and pull requests which are requests contributors can make to merge their code changes with the source of truth. The master branch is used as the source of truth.

5.2 Iterative design

The development of the service was done using the iterative design methodology. Based on the methodology, early on in the development process a prototype was made which was then updated on a regular basis. The updated prototype was used to ask for feedback, this ensured that there were many points during the development process where the clients could give feedback and had a saying in what the next steps should be. It also reduced time wasted on things that would not work out in the end because they could be spotted early on.

The following list specifies the iterations that were done to develop the service. The list is simplified and only includes notable iterations to give a sense of how the development process went.

1. Minimal dCGP example showing a random chromosome.
2. Add an evolution view.
3. Implement start, stop and pause of the evolution.
4. Add a plot for the loss of the evolution over time.
5. Redesign the UI.
6. Add active kernel selection functionality.
7. Add a plot for the labels of the data set and predictions of the expression.
8. Add network size and connection selection functionality.
9. Add data set presets.
10. Add the prediction equation view.
11. Add constants functionality.
12. Add gradient descent algorithm option.

[12] CRA can not update when the eject command is executed.

5.3 Cases

This section will describe three interesting and challenging implementation cases.

5.3.1 UI design

The UI of the application was created in three iterations. The first iteration naively added all the functionalities of the service on the page. The second iteration managed to put all the settings in the initial view of the page to remove the need for users to navigate around to start evolving. All the settings are by default set to a valid configuration that is able to evolve so new users are able to press start and the service will try to evolve to the optimal solution right away. The second iteration made sure that the second point stated in section 3.1.1, about approachable UI, is met by using industry standard UI design principles for user input elements like radio buttons, checkboxes and icon buttons.

The final design shown in figure 5.1 has more functionalities and rearranges the settings panels to mimic the steps early adopters during validation naturally took (Heddes, 2019b). Most users would first select their network size and whether they want constants. Next they would decide which kernels can be used. Lastly, the user would switch between evolving and changing the algorithm which are therefore placed next to each other at the end of the row. The figures showing the design iterations can be found in appendix B.

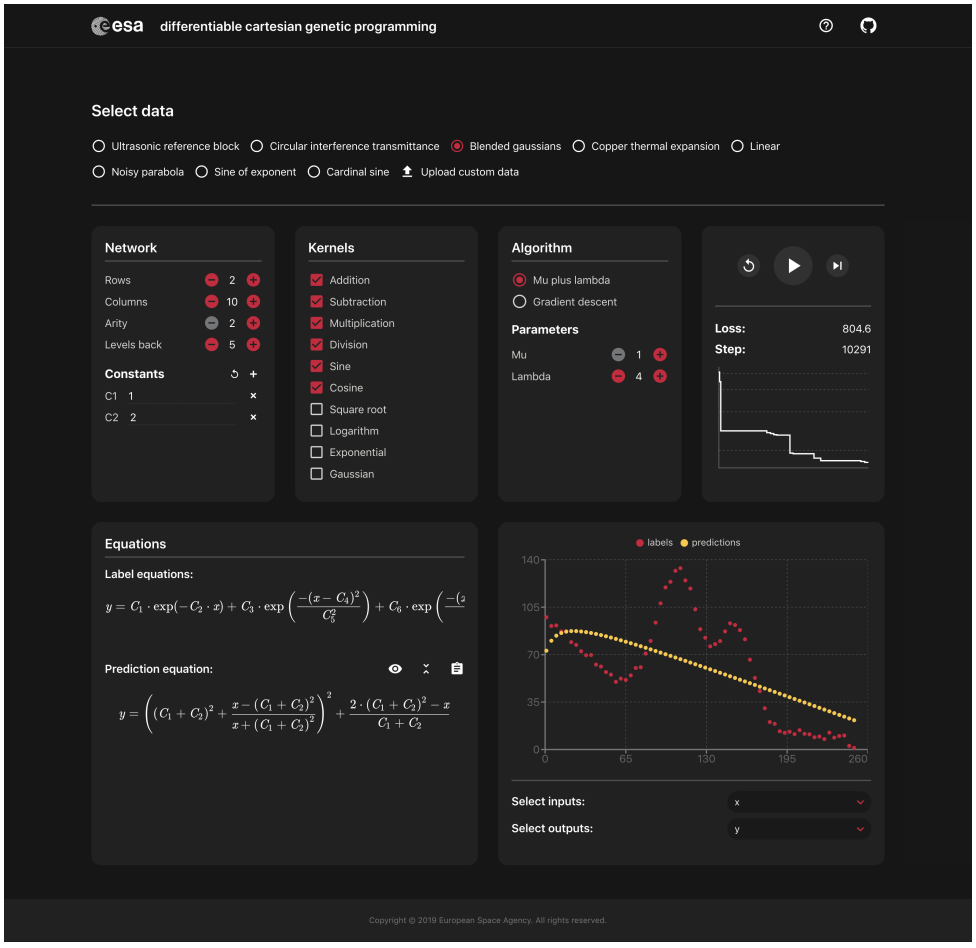

Figure 5.1: dCGP service final UI design.

5.3.2 WebAssembly bindings

The dCGP library depends on 6 other C++ libraries. In order to compile the dCGP bindings to WebAssembly these dependent libraries needed to be compiled using Emscripten. The dCGP bindings were

then compiled and linked against the compiled dependencies to create the dcgp.wasm WebAssembly file which can be loaded in JavaScript to use dCGP. The API of the dCGP WebAssembly bindings should be clear and familiar for developers who have used the Python or C++ dCGP library. However, with the generated dcgp.wasm file alone this was not the case. This is because the bindings require that complex parameters are encoded and stored in shared memory between WebAssembly and JavaScript which makes the API hard to understand. To improve this a new library called dcgp.js was created. Users of the dcgp.js library do not need to worry about WebAssembly or the bindings. This makes the API of the library feel as if it is using solely JavaScript. The library encodes and decodes the arguments and passes them on to the WebAssembly C++ code where the low level execution happens. A detailed explanation of the WebAssembly bindings can be found in appendix C.

5.3.3 Web Worker

To prevent the execution of dCGP from blocking the JavaScript main thread, which must do the UI updates, the dCGP code is executed in a separate thread. In JavaScript a separate execution thread can be created with a Web Worker. The messages to and from the dCGP thread are going through a proxy[13] which makes interacting with the dCGP thread similar to other asynchronous functions in JavaScript. The proxy sits on the main thread instead of on the dCGP thread because it tries to simplify the requests to and responses from the dCGP thread that are invoked on the main thread. The dCGP thread doesn't need a proxy because its only concern is knowing what is requested and responding accordingly. The proxy keeps track of which response from the dCGP thread corresponds to which request from the main thread. The reason why a proxy is used to centralize the communication between the threads is covered in appendix E.2.

On the dCGP thread requests are handled using Observables which can be seen as streams. A request comes in as an item in the stream and based on the type of request it gets transformed, handled and at the end piped into the responses of the worker. Observables are used because of its ability to describe the flow of data over time whereas other methods only describe the data flow at a given moment. A detailed explanation of the use of Web Workers in the application can be found in appendix D.

5.4 GitHub repositories

The result of this project is spread over two GitHub repositories which are called dcgp-web[14] and dcgp.js[15]. There are three main software development platforms, besides GitHub there are also GitLab and Bitbucket. All three offer mostly the same functionality but GitHub is the most popular as seen in appendix F which makes projects hosted on GitHub more likely to be discovered. ESA has an organization page on GitHub for all its open source projects. Based on these points it was decided to use GitHub as the development platforms for this project.

dcgp.js is the stand-alone JavaScript library for the C++ bindings of the dCGP library. dcgp-web contains the code for the website and uses the dcgp.js library for its calculations. Spreading the project over two repositories adds a distinct separation between the library and the website. This helps future maintainers and contributors to find what they are looking for faster. It also allows contributors that are not interested in the website to work on the dcgp.js library without needing to interact with the website aspect of the service and vice versa.

5.5 Tests and validation

The dcgp.js library has unit tests for every function and for some extreme equation cases to prove that the library is functioning correctly and to prevent regression from happening. All the tests for dcgp.js are automatically run on every pull request on GitHub, this is part of the continuous integration setup for the project. If the tests do not pass the pull request is not allowed to be merged with the code base.

[13] A proxy is an intermediary for requests and responses.
[14] dcgp-web repository: https://github.com/esa/dcgp-web.
[15] dcgp.js repository: https://github.com/esa/dcgp.js.

This prevents the source of truth from containing regressions and lowers the change of having bugs. A more in-depth description of unit tests and an example are given in appendix G.

The web application has continuously been tested and validated by members of the ACT (Heddes, 2019b) who would report issues which would then be fixed for the next iteration. The web application also has unit tests to determine whether the application can render without errors and to verify that the application meets the requirements. These tests prevent a pull request from merging that does not meet the requirements. In appendix E two bugs that were discovered during the development phase are discussed. In appendix H three manual verification cases are covered that show that the application is handling these cases correctly.

5.6 Documentation

Because the dcgp.js library can be used in other projects by different developers it is fully documented[16]. The expression page of the documentation is shown in figure 5.2 in which the parameters for an expression are stated with a description and an example of how to create an expression. The documentation is similar to that of the Python dCGP documentation. All the helper functions that are not exposed to the users of the library are documented using structured comments called JSDoc which has become the standard way of documenting in JavaScript. The comment describes the function, its parameters and what it returns. This is helpful information for the next developer that needs to work with this function or wants to use it.

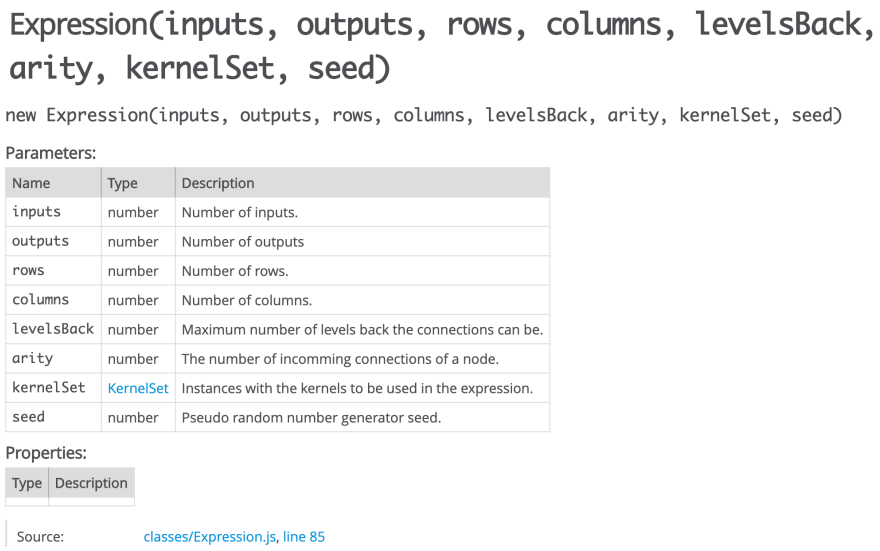

Figure 5.2: dcgp.js expression documentation page.

For both the dcgp.js library and the web application the code is as self documenting as possible by choosing appropriate variable names and using a clear and consistent code style. Self documenting code can be seen as the opposite of minified code in which all the variable names are as short as possible to make the file as small as possible which compromises the readability of the code. Dividing the code into functions with a distinct responsibility also improves the readability of the code. Prettier[17] is used to achieve a consistent code style because of its wide adoption in the JavaScript community as seen on the users page of their website[18].

[16] dcgp.js documentation page: https://esa.github.io/dcgp.js/.
[17] Prettier website: https://prettier.io/.
[18] Prettier users page: https://prettier.io/en/users/.

6 Realizing

This chapter describes the final steps in realizing the service covering distribution, marketing and maintenance.

6.1 Distribution

The distribution happens via GitHub's servers because they offer a free solution for serving static files. Everyone can navigate to the website[19] in their browser of choice. The GitHub servers use a large content delivery network with locations across the world. This means that the web application is served reliably and with low latency because there is almost always a data center nearby.

Because of the continuous integration setup a new version of the service will be released automatically every time someone makes a changes to the code on the master branch of the project on GitHub. This new version will then automatically be served to new visitors of the service and existing users of the service will get the update on the next visit[20].

6.2 Marketing

The ACT will promote and highlight the service on their website[21] and members of the ACT will demonstrate the tool to colleagues and other people with an interest in dCGP. For instance, Dr. Märtens told his colleagues from a different department about the dCGP service and Dr. Summerer, the head of the ACT, shared the dCGP service with Dr. Franco, the Director of Technology of ESA, who then shared the service with the departments that might be interested. In addition, the service will be announced on genetic programming and web development related blogs and forums.

6.3 Maintenance

Because the entire web application is compiled to static files there should be little to no maintenance necessary but in the occasion that the service needs to be updated Dr. Märtens will lead the task and assign someone to make the appropriate changes. These changes will then be merged with the GitHub repository if all the tests passed a new version of the application will automatically be released because of the continuous integration setup.

[19] dCGP web application https://esa.github.io/dcgp-web/.

[20] Because the service uses a web technology called service workers to cache resources, which drastically decreases page load time, the new version of the service will be served to existing users of the service after they closed all their browser tabs of the service.

[21] ACT website: http://www.esa.int/gsp/ACT/index.html.

7 Conclusion

The goal of this project was to make it more convenient to use dCGP and to make tools that provide insight in the evolution and the resulting expression and equations. These two goals were grouped under the phrase "providing dCGP as a service". The underlying idea was to move dCGP one step further from an academic experiment towards a product (software tool) helpful for engineering. The main question to answer during this project was *how can one best provide dCGP as a service.*

dCGP is a framework that finds a mapping between numerical data in the form of an equation. This works by combining the dCGP library with an evolution algorithm that will change the encoded values of the expression. This approach can be used for any problem where the mapping between input and output data is of importance. The analysis showed that mainly scientists in the field of genetic programming and engineers in general are interested in a dCGP web application to simplify the use of dCGP.

To provide an useful dCGP service for scientists and engineers analysis showed that it needs to be able to run an evolution algorithm, change the parameters of the evolution and the expression and, users must be able to upload custom data. The C++ dCGP library is integrated in the service by compiling the library to WebAssembly, this way code executes at near native speed. The user interface of the application is made using React because it has many resources and large developer and corporate backing.

Using dCGP is now effortless for every engineer, simply by navigating to the website of the service, uploading data and evolving. This opens up the possibilities for many projects to use dCGP to do analysis on data. Genetic programming scientists can now use the service to quickly test their cases and to get an intuition of how an evolution behaves over time.

8 Recommendations

To provide users with a better intuition on how the graph structure of dCGP works a view of the graph can be added to the application by the maintainers and/or contributors of the project. The graph view can show which kernels are used and which nodes are connected. This could change in real-time while the application is evolving so the user can see how the graph structure changes over time. Each node could show its value so it is easier to reason about how the output is formed based on the intermediate values.

The graph view can be implemented by making a grid view with the same number of rows and columns as the expression's cartesian grid. Each cell of this grid represents a node of the computational graph. Each node has inputs and a kernel type associated to it. When starting from the outputs all the connections can be traced by following the inputs of every visited node. With this implementation the unvisited nodes are inactive, meaning they do not contribute to an output, and can get a distinct style.

In addition to the $ES - (\mu + \lambda)$ and gradient descent algorithms an algorithm combining both can be added by the maintainers and/or contributors of the project to achieve potentially better results in a shorter time. The hybrid algorithm should first mutate the current μ best chromosomes to get λ mutated chromosomes. Then, before selecting the new μ best chromosomes perform N amount of gradient descent steps on the λ mutated chromosomes. This algorithm will improve the evolution for the cases where the mutation makes an equation similar to the target mapping but the provided constants have values far from the target which results in that mutation not being selected.

References

Google. (2019, June 19). Software development platform interest over time [Google trends]. Retrieved June 19, 2019, from https://trends.google.com/trends/explore?cat=5&q=%2Fm%2F04g0kcw,%2Fm%2F0125_4f0,%2Fm%2F05mx6p6

Green, B., & Seshadri, S. (2013, April 8). *AngularJS*. O'Reilly Media, Inc.

Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., ... Bastien, J. (2017). Bringing the web up to speed with WebAssembly. In *Proceedings of the 38th ACM SIGPLAN conference on programming language design and implementation* (pp. 185–200). PLDI 2017. event-place: Barcelona, Spain. doi:10.1145/3062341.3062363

Hamill, P. (2004, November 2). *Unit test frameworks: Tools for high-quality software development*. O'Reilly Media, Inc.

Heddes, M. (2019a, February 28). dCGP service, users and functionality. Interview with Dr. M. Märtens research fellow in the Advanced Concepts Team of the European Space Agency.

Heddes, M. (2019b, April 16). dCGP web service feedback. Interview with E. Ozturk young graduate trainee in the Advanced Concepts Team of the European Space Agency.

Heddes, M. (2019c, February 4). Introduction to dCGP as a service. Interview with Dr. M. Märtens research fellow in the Advanced Concepts Team of the European Space Agency.

Izzo, D., Biscani, F., & Mereta, A. (2016, November 15). Differentiable genetic programming. *arXiv:1611.04766 [cs]*. arXiv: 1611.04766. Retrieved January 31, 2019, from http://arxiv.org/abs/1611.04766

Miller, J. F., & Thomson, P. (2000). Cartesian genetic programming. In R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, & T. C. Fogarty (Eds.), *Proceedings of the 3rd european conference on genetic programming* (pp. 121–132). Lecture Notes in Computer Science. doi:10.1007/978-3-540-46239-2_9

Occhino, T., & Walke, J. (2013, July 8). *Introduction to react.js*. Seattle, WA, USA. Retrieved June 19, 2019, from https://youtu.be/XxVg_s8xAms

Turner, A. J. (2015, September). *Evolving artificial neural networks using cartesian genetic programming*. University of York. Retrieved from http://etheses.whiterose.ac.uk/12035/1/thesis.pdf

You, E. (2017, August 29). *Vue.js: The progressive framework*. New York, NY, USA. Retrieved June 19, 2019, from https://youtu.be/p2P3z7p_zTI

Zakai, A. (2011). Emscripten: An LLVM-to-JavaScript compiler. In *Proceedings of the ACM international conference companion on object oriented programming systems languages and applications companion* (pp. 301–312). OOPSLA '11. event-place: Portland, Oregon, USA. doi:10.1145/2048147.2048224

Appendices

A Differential Cartesian Genetic Programming **22**

A.1 What dCGP does . . . 22

A.2 How dCGP works . . . 22

A.3 What dCGP can be used for . . . 22

B UI design **24**

C WebAssembly bindings **27**

C.1 Dependency compilation . . . 27

C.2 Provided API . . . 27

C.3 Memory management . . . 28

D Web Worker system **29**

D.1 Proxy . . . 29

D.2 Observables . . . 29

D.3 Consistent framerate . . . 30

E Bugs **31**

E.1 Protected division . . . 31

E.2 Evolution loop . . . 31

F Software development platform interest **32**

G Unit tests **33**

H Service validation **34**

I Interviews **37**

I.1 Introduction to dCGP as a service . . . 37

I.2 dCGP service, users and functionality . . . 37

I.3 dCGP web service feedback . . . 37

A Differential Cartesian Genetic Programming

This appendix covers what dCGP is, how it works and what it can be used for.

A.1 What dCGP does

The dCGP library provides three expressions, an unweighted expression, a weighted expression and an Artificial Neural Network (ANN) expression. These three expressions encode a directed acyclic graph of computational nodes as a chromosome. In *Evolving Artificial Neural Networks using Cartesian Genetic Programming* Turner describes the chromosome encoding used in CGP which is equivalent to an unweighted expression (2015, para. 3.2). CGP was originally presented in *Cartesian Genetic Programming* (Miller & Thomson, 2000) as a new form of Genetic Programming. The weighted expression in dCGP inherits from the unweighted expression and adds a weight for every connection in the network. The original dCGP paper includes an illustration of a weighted network (see Izzo et al., 2016, fig. 2). The ANN expression inherits from the weighted expression and adds a bias for every node in the network.

When the dCGP library is combined with AuDi[22], a Automated Differentiation (AuDi) library, it provides a way of getting automatic higher order derivatives from the dCGP expression. This combination makes it possible to learn the constants of a function without needing to feed ephemeral constants[23] in the weighted or ANN expression. Feeding ephemeral constants in an unweighted expression was the standard approach prior to dCGP. Algorithms to evolve a dCGP expression are not part of the dCGP library and need to be made by the user of the library. A commonly used algorithm to evolve chromosomes is the $ES-(\mu+\lambda)$[24] algorithm which keeps the best chromosome, copies the μ best chromosomes to the λ others and mutates all λ chromosomes.

A.2 How dCGP works

When initializing a new dCGP expression the connections, kernels and if applicable the weights are randomly picked between a lower and upper bound which are determined by the hyperparameters. When one decides to perform n mutations on the chromosome n random indices of the chromosome are selected and are randomly reassigned between their bounds. In addition to performing point mutations one can also choose to perform more specific mutations, for example, on active nodes only.

When using dCGP[25] in combination with AuDi the function that is encoded in the chromosome is expressed in generalized dual numbers (gdual) instead of plain C++ double floating point numbers. AuDi transforms the encoded function into a Taylor expansion around a point which allows to compute the value of the function and all derivatives up to the order of the encoded function. The mathematics and examples of using truncated Taylor polynomials are covered in the original dCGP paper (Izzo et al., 2016), code examples are provided on the AuDi and dCGP documentation website.

A.3 What dCGP can be used for

dCGP can be used to evolve a chromosome that encodes the mapping between any number of inputs and outputs. The only requirement is that a function must be provided that can rank the chromosomes according to their accuracy. This function typically calculates the difference between labeled data, which is used as a reference, and the predictions. An example of such a formula is the root mean squared error (RMSE) as seen in equation 1 where T are the number of predictions, $\hat{y}$ are the labels and y are the predictions.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}} \tag{1}$$

[22] AuDi documentation: https://darioizzo.github.io/audi/.

[23] Ephemeral constants are constants that are guessed by the user with the idea that the chromosome will evolve the constant to get the correct value.

[24] Read as *evolution strategy mu plus lambda*.

[25] dCGP documentation: https://darioizzo.github.io/dcgp/.

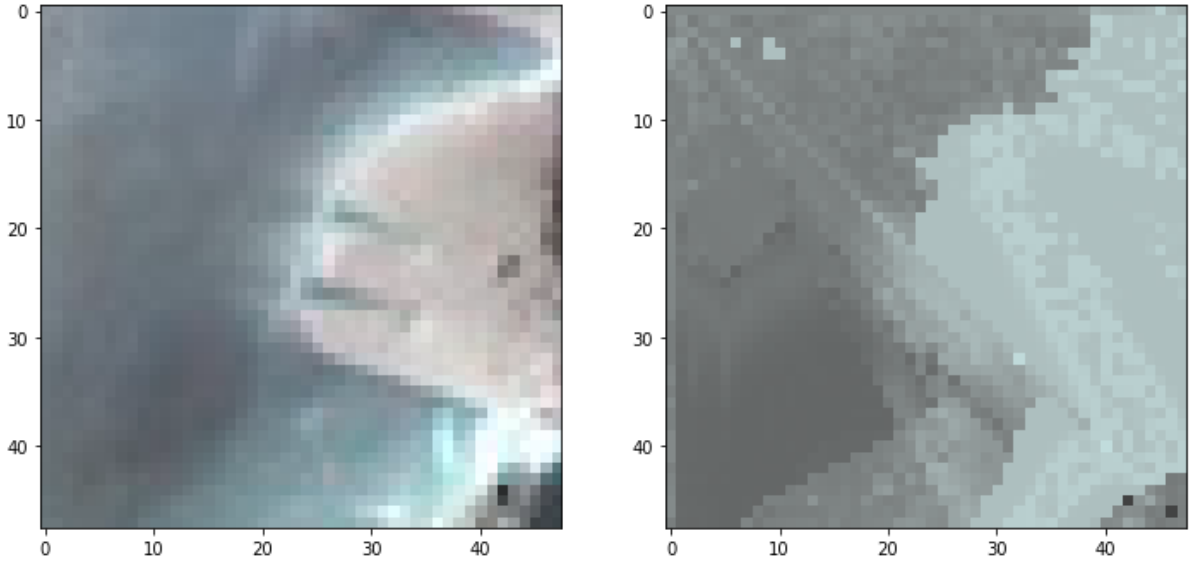

Figure A.1: Left shows the input image which was used as the labels. The original image is titled *ExoMars images Korolev Crater* © ESA/Roscosmos/CaSSIS and is a view of the rim of Korolev crater on Mars. Right shows the result of the evolution. For this encoding an unweighted dCGP expression was used with $r = 4$, $c = 512$, $l = 32$, $a = 2$, $N_{in} = 3$, $N_{out} = 3$. The kernel functions $+$, $-$, $*$, $\div$, sin, cos and ln were used.

Use cases of dCGP include optimizing trajectories, finding the relation between material properties, optimizing schedules, robot controllers and image classifiers. Figure A.1 shows the result of an evolution that tried to find the mapping between coordinates and colors. The dCPG expression used during evolution has 3 inputs, the first two are the x and y coordinates and the third input is an ephemeral constant with value 1. The expression has 3 outputs which are the red, green and blue color channels of the pixel at location x, y. Although the exact representation of the reference image was not found it is clear that the evolution found some features that are present in the reference image. A larger weighted dCGP expression is likely to improve these results.

B UI design

This appendix contains the design iterations for the UI of the service.

The UI of the application was created in three iterations. The first iteration, as seen in figure B.1, naively added all the functionalities of the service on the page. The second iteration, as seen in figure B.2, managed to put all the settings in the initial view of the page to remove the need for users to navigate around to start evolving. All the settings are by default set to a valid configuration that is able to evolve so new users are able to press start and the service will try to evolve to the optimal solution right away. The final design, as seen in figure B.3, has more functionalities and rearranged the setting panels to mimic the steps early adopters during validation naturally took (Heddes, 2019b). Most users would first select their network size and whether they want constants. Next they would decide which kernels can be used. Lastly, the user would switch between evolving and changing the algorithm which are therefore placed next to each other at the end of the row.

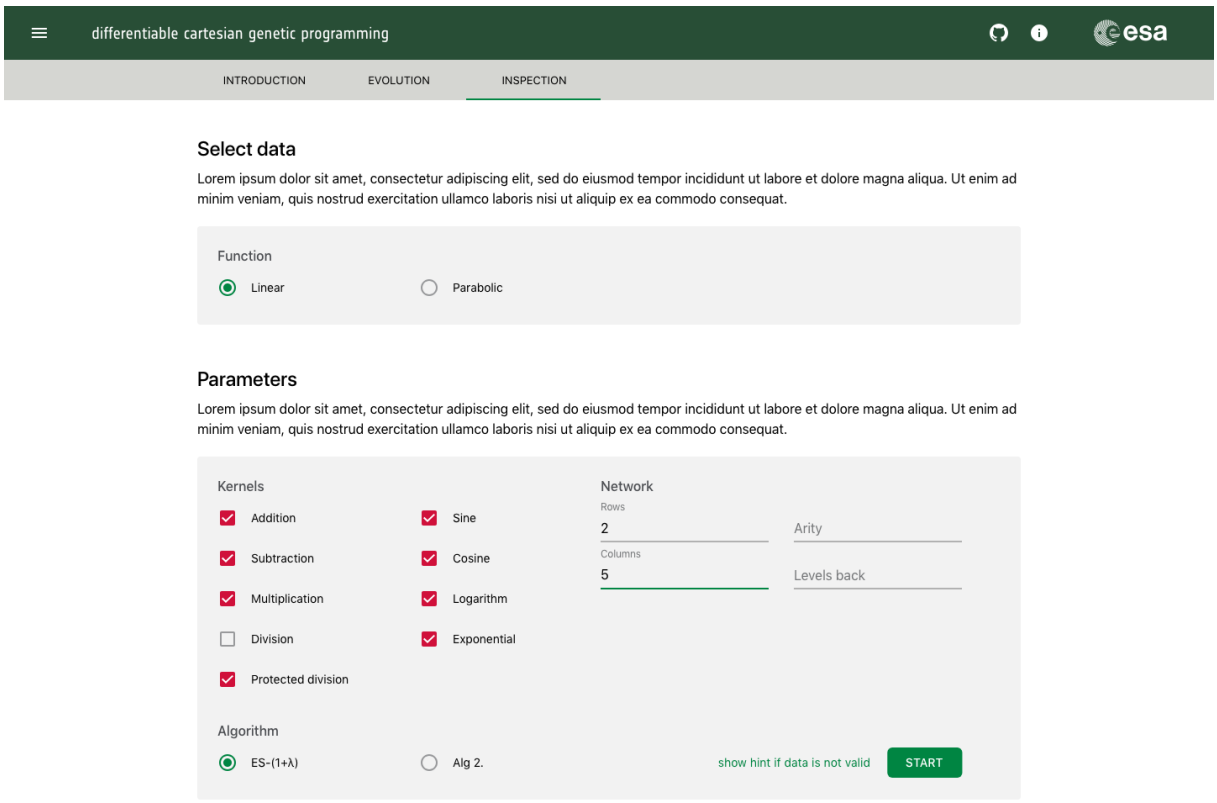

Figure B.1: dCGP service UI design iteration one.

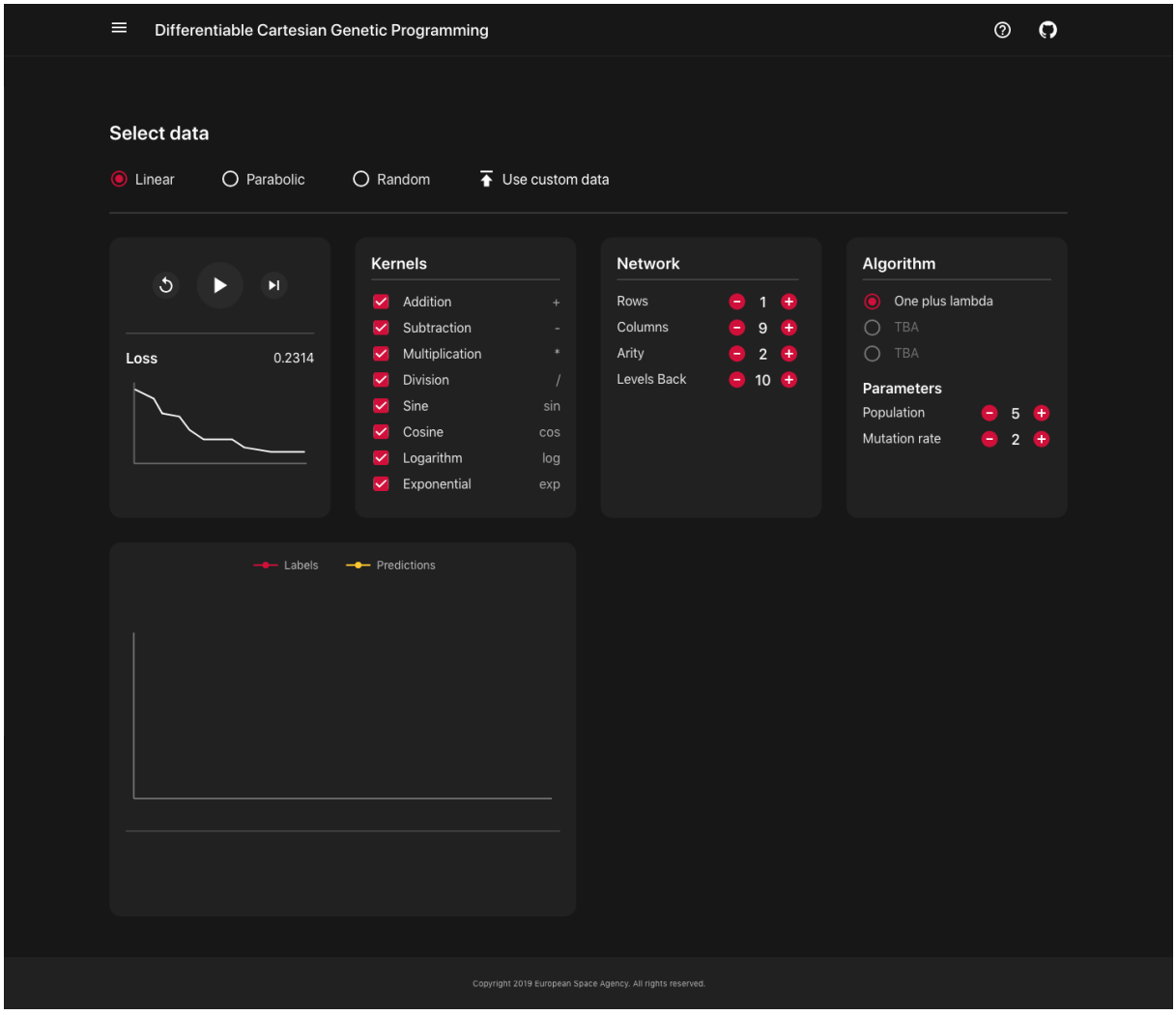

Figure B.2: dCGP service UI design iteration two.

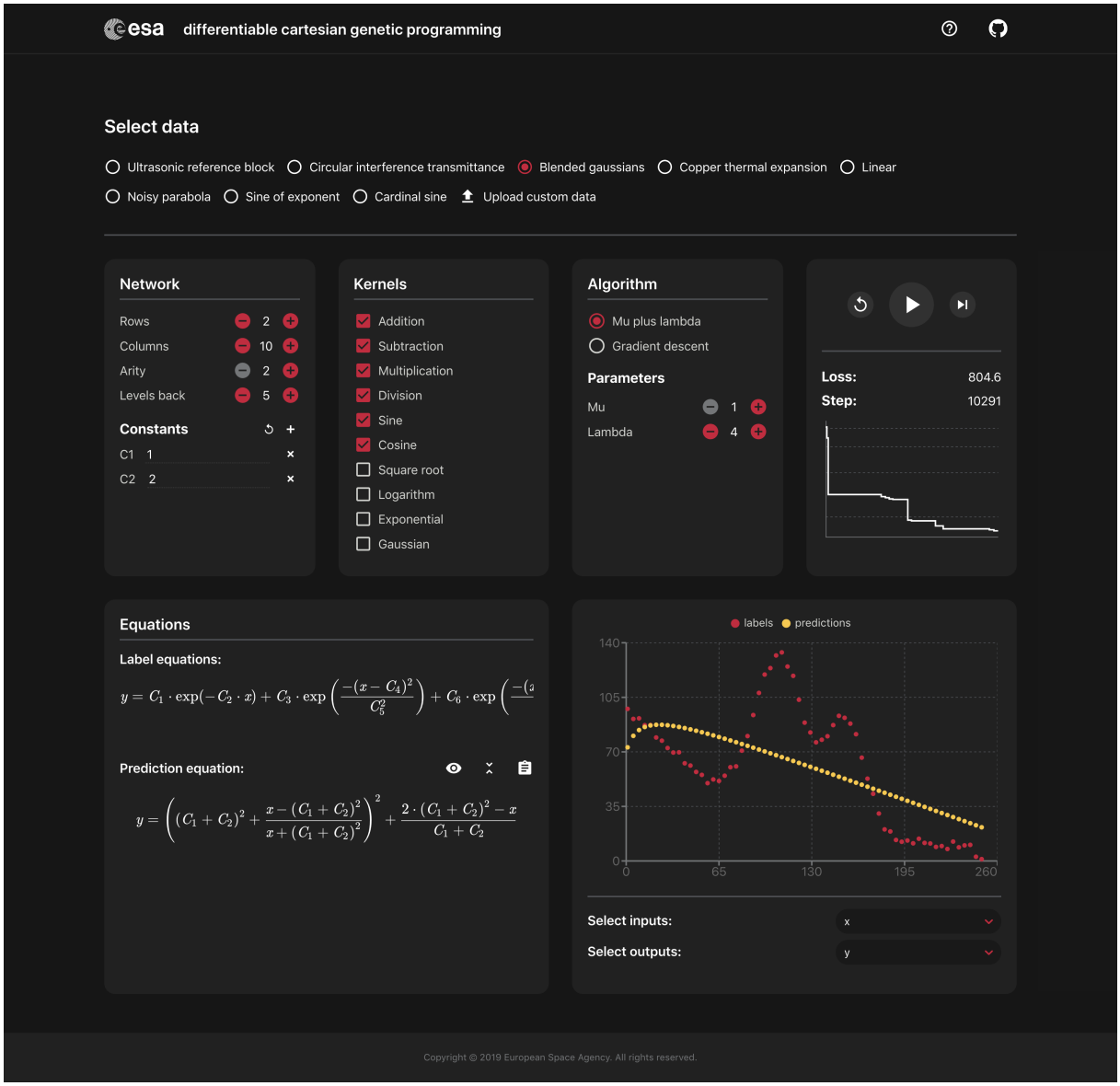

Figure B.3: dCGP service UI design iteration three.

C WebAssembly bindings

This appendix describes the WebAssembly bindings implementation. The WebAssembly bindings are the to WebAssembly compiled C++ functions that interact with the dCGP library. The generated WebAssembly file can be loaded in JavaScript which is then able to call the exported C++ functions.

C.1 Dependency compilation

In order to be able to compile the dCGP C++ library all of dCGP's dependencies need to be compiled first, which are:

- Boost version 1.6.1.
- GMP version 6.1.2
- MPFR version 4.0.1
- MP++ version 0.9
- Piranha version 0.11
- AuDi version 1.6.3
- dCGP version 1.2.0

The installation scripts for Boost, GMP and MPFR are based on a project by Marcos Scriven called CGAL[26] in which he compiled these libraries using Emscripten. The complete compile and install script can be found on GitHub as a Dockerfile[27]. The pseudo-code for installing the dependencies can be found in algorithm 1. The related discussions are found in issue 10 on GitHub[28].

Algorithm 1: Install dCGP's dependencies

Result: Compiled and installed dependencies

```
for each dependency do
    Download sourcecode;
    Checkout at specific version;
    Configure build script;
    Compile and install;
end
```

C.2 Provided API

The exported C++ functions can only take numbers as arguments and may return a single number. This restriction makes working with the raw exported functions hard and leads to an unpleasant developer experience. Due to the fairly recent introduction of WebAssembly transferring complex data is not implemented in the WebAssembly specification yet but is on the roadmap for the coming years[29]. This means that for the time being all the higher level datatypes such as strings, arrays and objects need to be encoded in numbers only before being provided as arguments to the exported functions.

The details on how to encode and decode common JavaScript data structures is covered in section C.3. To remove the overhead of using the raw exported C++ functions a wrapper library was made called dcgp.js[30] to provide an easy to use abstraction over the raw C++ function exports. The API implementation[31] is based on the Python implementation of dCGP called dcgpy[32] but makes use of property getters and setters in JavaScript to make the API follow the commonly used JavaScript coding style.

[26] CGAL repository: https://github.com/marcosscriven/cgaljs

[27] dCGP dependencies installation script: https://github.com/mikeheddes/dcgp.js-dependencies-image/blob/master/Dockerfile

[28] Installation discussion on GitHub: https://github.com/esa/dcgp.js/issues/10

[29] WebAssembly future features: https://webassembly.org/docs/future-features/.

[30] dcgp.js repository: https://github.com/esa/dcgp.js.

[31] dcgp.js documentation: https://esa.github.io/dcgp.js/.

[32] dcgpy documentation: https://darioizzo.github.io/dcgp/docs/python_docs.html.

dcgp.js handles encoding the arguments, decoding the result and provides more readable errors when incorrect arguments are provided. The users of dcgp.js now only have to think about ways to use the higher level API instead of needing to think of how to encode and decode the data for every call to dCGP. This essentially makes the entry level to dCGP in JavaScript lower. In addition dcgp.js can both be used on the server and on the web which broadens the scope of developers and projects which can use dCGP.

C.3 Memory management

Instantiating a WebAssembly file in JavaScript results in a memory object which is a JavaScript ArrayBuffer. This memory object is the shared memory between the WebAssembly and JavaScript environment which both environments can read from and write to. The memory can be represented in any supported number type as a TypedArray, these include:

- `Int8Array` for a signed 8 bit integer representation.
- `Uint8Array` for an unsigned 8 bit integer representation.
- `Int16Array` for a signed 16 bit integer representation.
- `Uint16Array` for an unsigned 16 bit integer representation.
- `Int32Array` for a signed 32 bit integer representation.
- `Uint32Array` for an unsigned 32 bit integer representation.
- `Float32Array` for a 32 bit floating point representation.
- `Float64Array` for a 64 bit floating point representation.

In C++ the types have different names but there is a well defined mapping between the C++ and JavaScript types. For instance, the C++ `double` type corresponds to the `Float64` JavaScript type and the C++ `unsigned int` type corresponds to the `Uint32` JavaScript type.

Both JavaScript and WebAssembly can access and manipulate the WebAssembly memory directly. This allows JavaScript to put the numbers representation of any data directly in the shared memory. The pointer to the memory location of the data together with some metadata of the encoded data are enough to decode the higher level data structure in C++. Emscripten provides a set of helper functions to reduce the overhead of allocating and restoring the shared memory. These functions are `stackSave`, `stackAlloc` and `stackRestore` which implements a stack-based memory management system.

A concrete example of how to encode a number matrix in pseudo-code is given in algorithm 2.

Algorithm 2: Encode a number matrix in JavaScript and place it in the shared memory.

input: Matrix X of size $w \times l$

```
// Get the linear values representation.
flat ← flatten(X);
// Transform the numbers into a specific typed array.
typed ← TypedArray(flat);
// Allocate the required number of bytes to store typed.
pointer ← stackAlloc(sizeof typed);
// Place typed in shared memory at the allocated location.
Set typed in memory at pointer;
```

The blog post by Marco Selvatici about WebAssembly[33] explains encoding, decoding and WebAssembly in general in greater detail with more examples and diagrams. In addition the blog posts by Lin Clark[34] who works at Mozilla (an initiator of WebAssembly) form a great resource to get a thorough understanding about how WebAssembly works.

[33] Marco Selvatici WebAssembly blog post: https://marcoselvatici.github.io/WASM_tutorial/index.html.
[34] Lin Clark blog: https://hacks.mozilla.org/author/lclarkmozilla-com/.

D Web Worker system

This appendix describes the Web Worker implementation. In JavaScript a Web Worker can be used to create a new thread in addition to the main thread. This thread can then be used to offload some of the work from the main thread to keep the UI, which must be computed on the main thread, responsive.

Initially all the interactions with dCGP were done on the main thread which resulted in a slow and sometimes freezing UI which led to a poor user experience. This was due to the evolution cycle which implements the start, pause, resume and reset functionality, as seen in algorithm 3. The evolution cycle was blocking UI updates from executing.

Algorithm 3: Evolution cycle.

Input: Event streams: `start`, `pause`, `resume` and `reset`.

```
1 On start event handleStart();
2 Function handleStart is
3     while evolving do
4         step();
5         yieldProgress();
6         handleNewEvents();
7     end
8 end
```

To improve user experience all the interactions with dCGP were placed in a separate thread. This however generated a different problem, due to the JavaScript Web Worker implementation[35] it is only possible to communicate between threads using the `postMessage` method to send messages and the `onmessage` event to read and handle incoming messages. With this implementation requests to the worker are not linked to responses from the worker which caused time inconsistencies as described in appendix E.2 and made for overly complicated code to handle the various response messages.

D.1 Proxy

These problems have been resolved by creating a proxy for the worker on the main thread that acts as a switch. The proxy adds a unique id property to every request which the worker reads and attaches to the response so the proxy knows the incoming message was the response for the request with the same id. For request-response type messages, which are the most common type, the proxy returns a Promise[36] that upon receiving the response to the request resolves with the by the worker returned value. For the evolution start method the proxy instead returns an Observable[37] because a single start request can result in multiple progress responses and a Promise can only be resolved once were as an Observable can emit multiple responses.

D.2 Observables

The dCGP thread needs to be responsive to updates from the main thread when updates for algorithm settings are provided. This for example happens when the user changes the algorithm type during evolution. It is difficult to develop a stable implementation using commonly used event handler functions because of the asynchronous nature of the incoming messages which resulted in difficult and hard to maintain code with time inconsistency bugs.

After using a proxy the remaining bugs were resolved by switching to Observable based logic in the dCGP thread. According to the Observable proposal “The Observable type can be used to model push-based

[35] JavaScript Web Worker documentation: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API

[36] JavaScript Promise documentation: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

[37] JavaScript Observable proposal: https://github.com/tc39/proposal-observable

data sources such as DOM events, timer intervals, and sockets." Because communication between threads is event based Observables are a natural way of handling incoming messages and because of their stream and pipe philosophy they are simpler to reason about.

D.3 Consistent framerate

The initial implementation of the evolution cycle used a hard-coded value for the amount of steps to evolve per evolution cycle. This resulted in an inconsistent framerate across devices and between different network sizes. In general this meant that with larger networks the UI would feel slower because the updates came in at the same step interval but not at the same time interval. This was improved by implementing an adaptive step size that would maintain a consistent time interval between UI updates.

E Bugs

This appendix covers two of the bugs that were discovered during the development phase of the dCGP service. The first bug was found in the protected division operator implementation and the second bug was due to time inconsistencies in the evolution loop.

E.1 Protected division

Dr. Izzo reported an issue where the displayed equation did not match the plotted graph on the service. After some time debugging the code for the service it became apparent that the issue would only occur when the protected division operator[38] was used in the generated equation. This raised the suspicion that there might be a bug in the implementation of the protected division operator or its print function[39].

The suspicion was correct, there was indeed a bug in the protected division operator and in its print function. Even though it was outside the scope of this project some time was spent to fix the protected division operator which led to a discussion with Dr. Izzo whether the protected division operator makes sense when used in a differential equation. There was mutually decided that the protected division should not be used when doing differential equations because the gradient that it creates does not represent what actually happens which can cause even harder bugs to occur. This decision led to the removal of the protected division from the service and restricted the protected division to only being used in a CGP type unweighted expression in the C++ dCGP implementation.

E.2 Evolution loop

As covered in appendix D the evolution cycle starts on the *start* event and loops until a *pause* or *reset* event. An issue with the evolution cycle was first discovered by Mr. Ozturk who reported an issue where the loss plot of the service would draw a loop-like shape. After some research it became apparent that the loop-like shape was caused by a new evolution progress event that happened after the evolution had already been paused or reset. Some more research showed that the new progress event was the result of a loose coupling between the dCGP thread and the main thread which caused these evolution progress events from not being discarded.

In order to resolve this issue a centralized communication point needed to be created for communication between the main and dCGP threads to remove the loose coupling that caused this bug. This point is called a proxy because it sits between the invoker (main thread) and the handler (dCGP thread) and all the requests and responses go through it. The proxy together with the use of Observables removed the time inconsistency by correctly discarding progress events after the evolution has been paused or reset. The implementation of the proxy and the use of Observables is covered in more detail in sections D.1 and D.2.

[38] The original dCGP implementation in C++ includes a protected version of the division operator. This protected version sets the result to 1 when dividing by 0 which would otherwise result in infinite numbers. Infinite numbers sometimes are preferably avoided because it can cause trouble in the calculations of evolution algorithms.

[39] In dCGP every operator has a associated print function which is called when the string representation of the operator is needed, for instance when the equation needs to be displayed.

F Software development platform interest

Figure F.1 shows the software development platform interest over time. Comparing the search interests for GitHub, GitLab and Bitbucket.

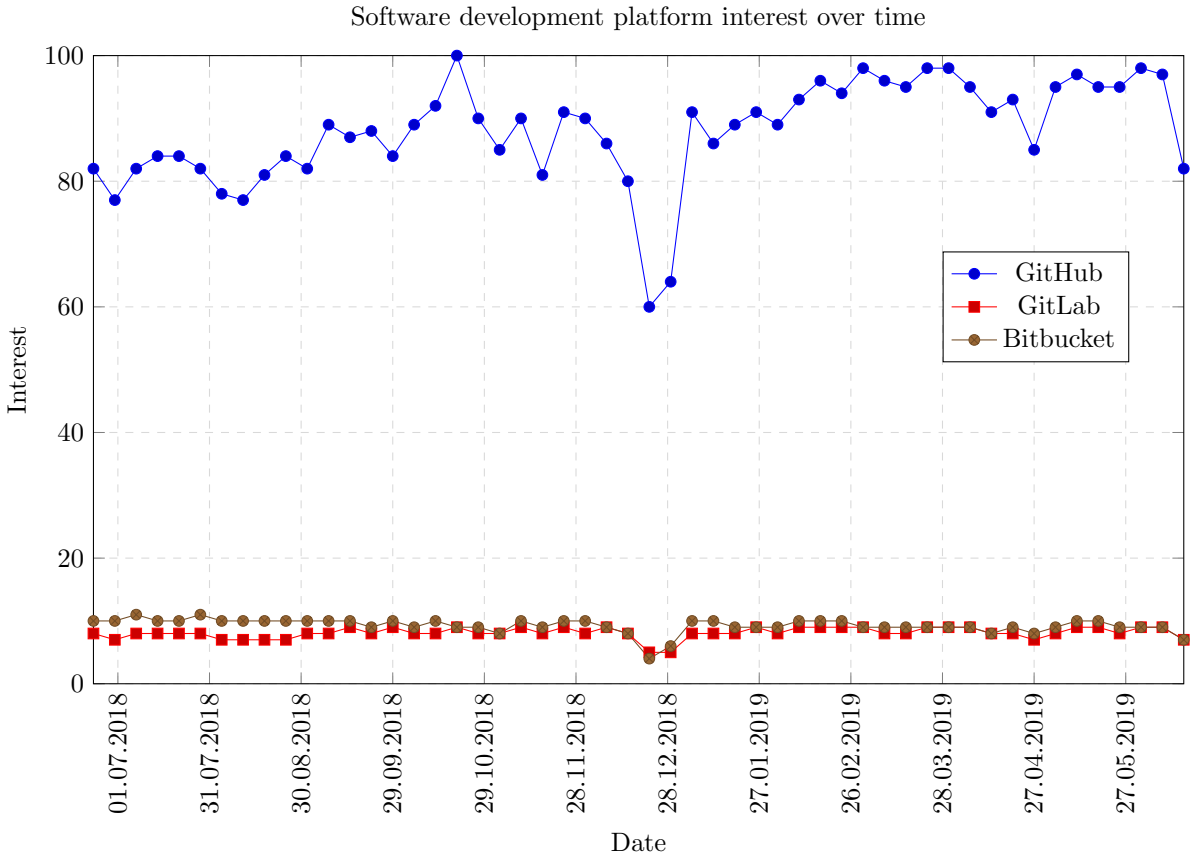

Figure F.1: Software development platform interest according to Google trends (2019). Numbers represent search interest relative to the highest point on the chart for the given time. A value of 100 is the peak popularity for the term. A value of 50 means that the term is half as popular.

G Unit tests

This appendix will cover the use of unit tests, what they are and how they work. Unit tests are typically automated tests written and run by software developers to ensure that a section of an application (known as the "unit") meets its design and behaves as intended (Hamill, 2004). The goal of unit testing is to isolate each part of the application and show that the individual parts are correct. A unit test provides a strict, written contract that the code must satisfy. If the actual behavior in a unit test deviates from the expected behavior the test will fail. Because unit tests are written in a programming language they can be integrated in any script. For instance, unit tests can be run as an automated step in a continues integration setup.

Listing 1 shows a basic example of a unit test taken from the dcgp.js source code. The test checks whether all the entries of the result of the `flatten2D` function match all the entries of the predefined `expectedArray`. It also checks whether the order of the entries is correct. The `matrix` variable defines a condition on which the `flatten2D` function, the unit of code, is being tested.

```
1  it('flattens a 2D array', () => {
2      const matrix = [[1, 2, 3], [4, 5, 6]]
3
4      const flat = flatten2D(matrix)
5      const expectedArray = [1, 2, 3, 4, 5, 6]
6
7      flat.forEach((value, index) => {
8          expect(value).toBe(expectedArray[index])
9      })
10 })
```

Listing 1: Unit test for a flatten array function.

The dcgp.js project has unit tests for all aspects of its code base. For example, it has unit tests for the mu plus lambda algorithm to check whether the expression after a number of evolution steps is improving and tests for the gradient descent algorithm to check whether the constants are being updated correctly. There is also a test that prevents regression from occurring by checking whether the loss after a number of gradient descent steps stayed the same over any number of runs of the unit test. This test makes sure that a change in a different function in the code base did not change the result of the gradient descent algorithm. There are also unit tests for a number of helper functions which range from encoding and decoding strings to flattening arrays as seen in listing 1.

H Service validation

This appendix shows three cases that were used to manually validate that the service is working correctly. In the first validation the cardinal sine function $\sin(x) \div x$ was fitted using the mu plus lambda algorithm without the sine or cosine kernels which resulted in an approximation of the function as seen in figure H.1. In the second validation the cardinal sine function was fitted using the mu plus lambda algorithm with the sine and cosine kernels both active. This setup found the exact solution as seen in figure H.2. In the third validation the cardinal sine function was fitted without the sine kernel and with an extra provided constant. First the mu plus lambda algorithm was evolving the expression followed by gradient descent which matched the target equation by setting the constant to half π as seen in figure H.3 which is equivalent to the offset of cosine and sine. These validations show the correct behavior for each setup.

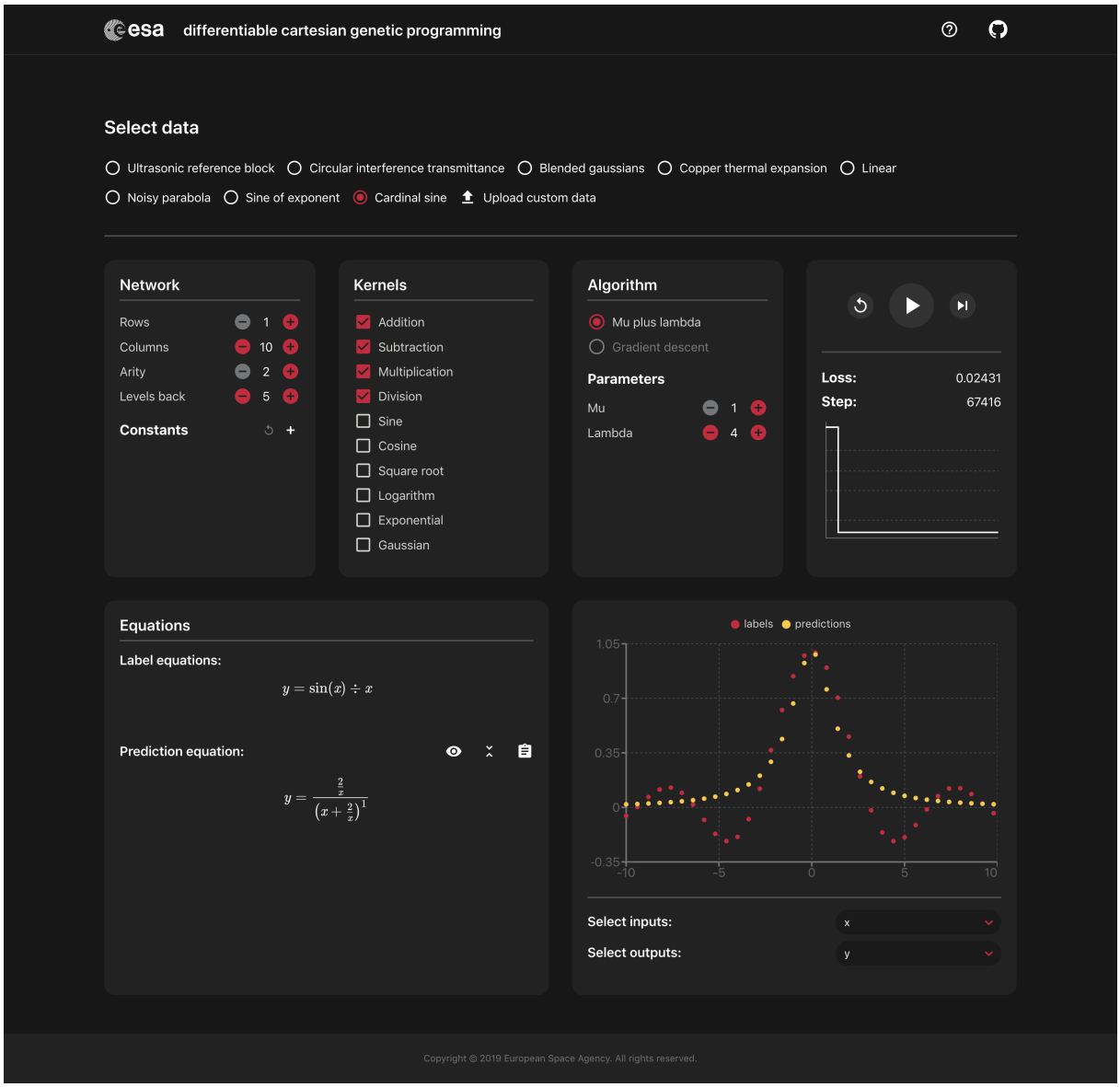

Figure H.1: dCGP service validation without the cosine and sine kernels.

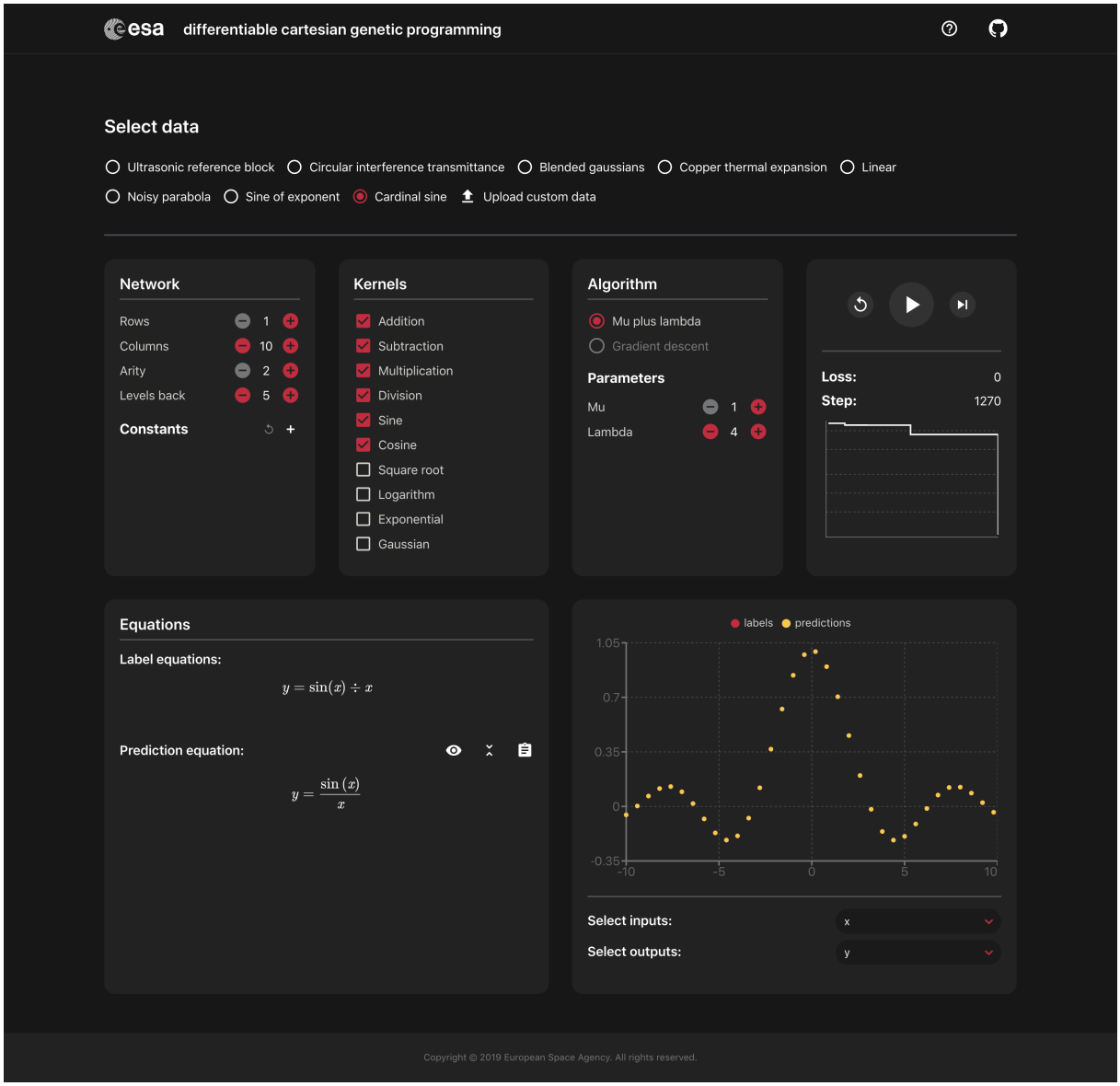

Figure H.2: dCGP service validation with the cosine and sine kernels.

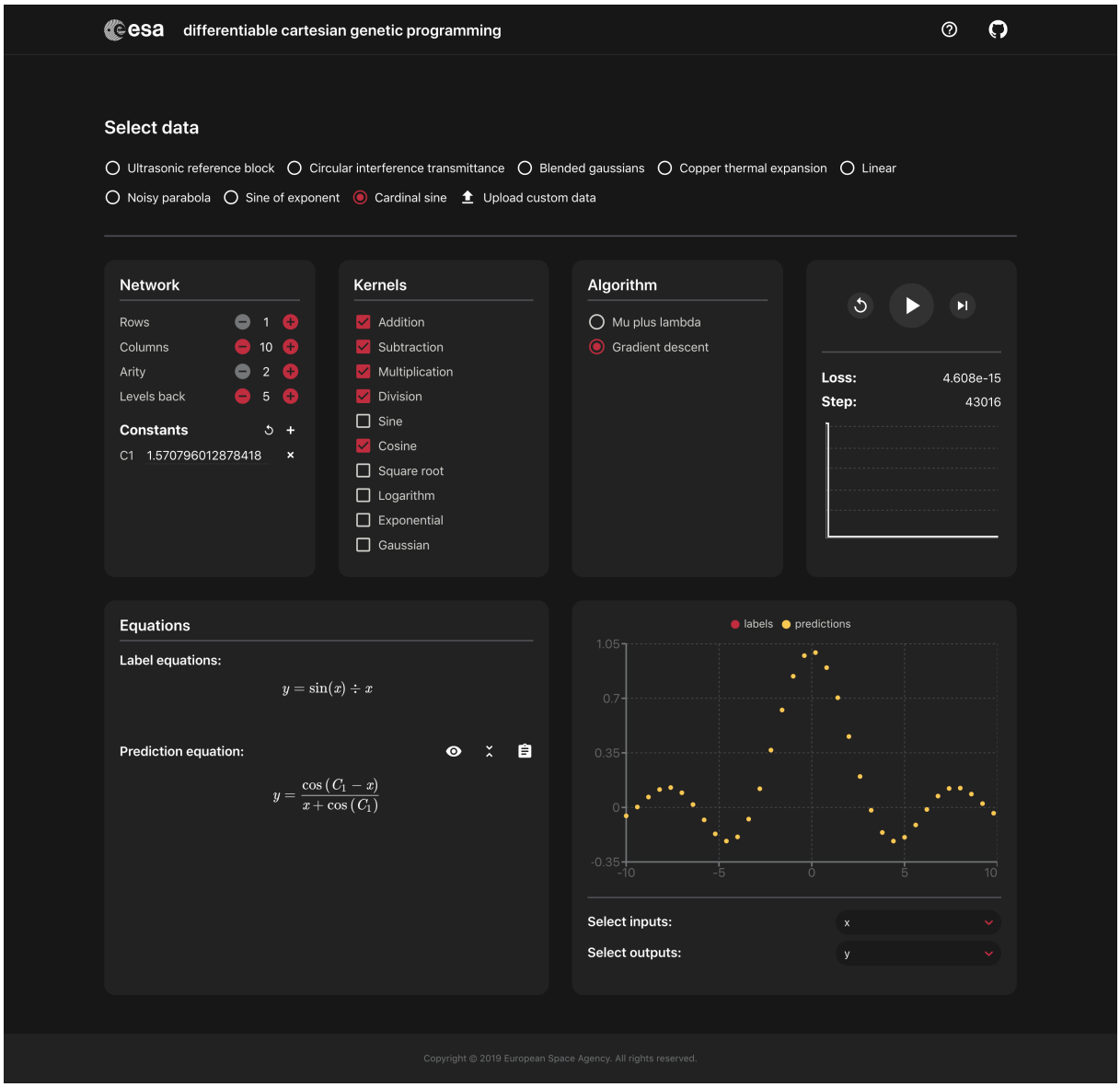

Figure H.3: dCGP service validation without the sine kernel and with a constant.

I Interviews

This appendix will cover three interviews that were conducted during this project. First Dr. Märtens was interviewed to get an understanding of dCGP and the goals for the dCGP service. The second interview with Dr. Märtens was regarding the implementation details of the service. The third interview was with Mr. Ozturk to get his feedback on the service.

I.1 Introduction to dCGP as a service

This section describes an interview with Dr. Märtens, a research fellow in the Advanced Concepts Team of the European Space Agency. This interview was conducted in the first week of the internship and was conducted to develop a feeling for the desired end result and to get up to speed with the dCGP implementation. In this interview Dr. Märtens was asked to explain how dCGP works, why dCGP was created and where to find resources to read up on dCGP. Lastly, he was asked why there is a need for the dCGP service. The details regarding how dCGP works can be found in appendix A, why dCGP was created is answered in the first paragraph of chapter 1. The resources Dr. Märtens provided are described in section 2.2. The need for a dCGP service is covered in sections 2.1, 3.3 and 2.4. This interview resulted in a good understanding of dCGP and developed a feeling for the desired end result.

I.2 dCGP service, users and functionality

This section describes the second interview with Dr. Märtens. This interview focused on the implementation details of the service. This interview was conducted one months into the internship. During this interview the features that the dCGP service should have were discussed together with the possible ways of developing the service and the technologies that could be used to build it. How Dr. Märtens envisioned the dCGP service is covered in sections 3.3 and 3.4. The technologies that can be used to develop the service are covered in chapter 3. This interview resulted in an overview of technologies which were further researched and were extended by new possibilities that where discovered during the research phase.

I.3 dCGP web service feedback

This section describes an interview with Mr. Ozturk, a young graduate trainee in the Advanced Concepts Team of the European Space Agency. This interview was conducted two and a half months into the internship and was meant to get feedback on the implementation of the service at the time. Mr. Ozturk mentioned that the order of the settings panels did not reflect the order that he would use them in, as discussed in appendix B, this behavior was afterwards verified by Dr. Märtens. In addition he discovered several bugs including the one covered in appendix E.2 and proposed several feature requests like being able to add negative constants and being able to evolve an initial expression that results in infinite numbers.