Google Summer of Code

Canadian Centre for Computational Genomics

Project Info

Project Title: Improved Tools for Genetic Diversity Modelling

Project Short Title: Optimizing Genetic Modelling

URL of project idea page:
C3G Project Idea List (see "Population genetics simulation and modelling")
Ryan Gutenkunst's ∂a∂i software

Biographical Information

I am an undergraduate at Duke University majoring in computer science and biology. I'm especially interested in the intersection of these two fields, so I am very excited at the possibility of working on a project in bioinformatics this summer. I have taken courses in a number of subjects that will aid me in completing the project this summer, including molecular biology, DNA science, algorithm design, discrete math, differential equations, and linear algebra. I am also fluent in both Python and C++, the two languages that will be used in the project.

In high-school, I participated in the USA Biology Olympiad, a competition that involves over 10,000 students across the country. After multiple rounds involving difficult theoretical questions and practical examinations, I placed first in the nation. I then advanced to the international competition in Bali with 240 students from 60 countries, and earned another gold medal. I also placed first internationally in the smaller University of Toronto Biology Exam. Additionally, I worked in a molecular genetics lab at the NIH one summer, where I studied the genetic basis to the development of the vertebrate vascular system, which involved performing DNA sequence analysis.

In computer science, most of my experience is academic. However, I performed an independent research study in biostatistics last semester, where I analyzed causes of depression, and its effects on performance in American adults. This involved using R to analyze a massive CDC dataset of health evaluations. I used a unique analytic approach, creating a structural equation model to set up relationships between different variables, and determine their correlations.

I believe I could be a great asset to this project, and hope to have the chance to contribute to it, as I will not only gain experience in a subject I'm passionate about, but will also provide software that can truly help the genetics community.

Contact Information

Name: Will Long

Postal Address:
College Address (Until May 7)
302 Gilbert-Addoms
1368 Campus Drive
Box 94878
Durham, NC 27708-4878

Home Address (After May 7)
1833 Nigel Court
Vienna, VA 22182

Cell Phone: (703) 965-1255

Email: wlong799@gmail.com

Other Communications:
Skype (wlong799@gmail.com)

Emergency Contact:
Diane Long (Mother)
(703) 965-0213
dianeslong@yahoo.com

Student Affiliation

Institution: Duke University

Program: Double majoring in biology and computer science

Stage of Completion: 1 Year

Verification Contact: Official transcript can be mailed/emailed upon request

Schedule Conflicts

None

Mentors

Mentor Name: Dr. Simon Gravel

Mentor Email: gravellab@gmail.com

After finishing my selection test, I emailed Dr. Gravel to introduce myself, and quickly explain my background and interest in the project. We then had a discussion over video chat, where he explained the possible projects for this summer, and asked me more about what I might be interested in and able to do. He followed up in an email earlier this week with the project he thought was best suited for me. We discussed the project some more, to make sure I understood the details and what was expected.

Synopsis

Interpreting genomic data largely relies on simulation software. One important family of software simulates population-level diversity over the course of evolution as a function of selection and demography. The ∂a∂i open-source package, developed by Ryan Gutenkunst (http://gutengroup.mcb.arizona.edu), is the leading software in the field, but is limited when handling large datasets.

This project aims to overhaul the computational engine of the ∂a∂i software by implementing a recently developed approach for solving partial differential equations. The project will use ∂a∂i's interface and code-base, but introduce a new spectrum simulation engine. Using a moment-based approach will improve performance and reduce numerical uncertainty, offering the genetics community with a general-purpose simulation tool that can address larger and more complex datasets than previously possible.

A prototype version of the computational engine is already running. The goal of this project will be to optimize the engine and embed it efficiently in the existing code-base.

Benefits to Community

The ∂a∂i open-source package has already greatly benefitted the genetics community, providing a flexible and relatively fast way to simulate population diversity over the course of evolution. Since its creation in 2009, it has become the leading software in its field, and has already been cited in nearly 300 published articles. These projects have included determining demographic histories, studying the origins of humans and their dispersal from Africa, identifying genes positively selected for by evolution, and more. Together, these projects have provided the community with much knowledge about evolution, the relationships between different species, and the distribution of fitness effects in genetic variants.

While ∂a∂i has clearly provided great benefits already, there are still improvements to be made. Average runtimes for a two-population model are about ten minutes, and more critically, it cannot model more than three populations, whereas many contemporary datasets contain dozens. Implementation of the new model will not only speed up runtimes and increase accuracy, but will allow for proper analysis of datasets previously difficult to deal with. In addition to providing the benefits listed above, it will help basic biomedical research aiming to understand the distribution of neutral and deleterious alleles in the human genome. This may eventually help in identifying disease risk alleles, which will allow prioritization of medical research efforts. Finally, the project will demonstrate the usefulness of the new PDE solution approach, which can be applied in a wide range of applications.

Coding Plan & Methods

There are four main parts to this project, listed below in mostly chronological order. Further details on project timing can be found in the Timeline section below. Step 1 is profiling, which will take place mainly at the beginning of the summer, but also throughout the project to ensure time-wasting code is optimized. The bulk of the project lies in step 2, which is where optimization of code for the new computational engine takes place. Step 3, improving the numerical optimizer, may be skipped in case an unseen issue comes up, and I run short on time. Step 4 is testing, and thus will be performed throughout the summer to ensure everything is running correctly.

1. Profile the code to find time wasting functions. In addition to the computationally expensive methods described below, there are a number of essential minor functions that are frequently called. Profiling the code will provide insight into which functions should be focused on to improve performance.
2. Refactor the Python engine and port to C++ as necessary to improve performance. Each simulation consists in the integration of a set of linear differential equations. This process can be made more efficient in a variety of ways.
    - For high-dimensional problems, generating the matrices necessary for the linear system can be computationally expensive. This step can be optimized by porting it to C++, or by keeping the matrices in memory for quick reuse in further simulations.
    - Resolving the set of differential equations is also very expensive. It consists in the resolution of a linear system ($Ax = B$). Matrix A is both sparse and banded, and with a few approximations, it can be written as a pentadiagonal matrix which is an especially useful form for numerical solutions. It may be possible to take advantage of this form, to code an efficient solver for resolution of pentadiagonal linear systems.
    - Optimize other functions as deemed necessary by the profiling step taken before. This may involve porting to C++ or making the code more efficient.
3. Improve the optimization algorithm used to compare simulations to data. Once the partial differential equation has been solved, it is possible to go from a set of model parameters to an experimental prediction. An algorithm must be used to find the parameters that best fit the data. Optimizing the code that does this can further improve the performance of the software.
4. Testing the code. This step will be performed throughout the summer to ensure code is working correctly. Two types of testing will be necessary throughout the summer: tests for accuracy (i.e. is the code working correctly), and tests for performance (i.e. is the code more efficient than the existing implementation).
    - In testing for accuracy, the *∂a∂*i package already provides a number of tests that can be used. Furthermore, as we can already find solutions using the existing *∂a∂*i software, it will be possible to compare the results from our new version against the old one. Thus, it is unlikely that new tests will need to be created for this purpose.

    - In testing performance, it will be necessary to have a variety of standardized data sets that I can run simulations on throughout the summer. As I gradually optimize the new computational engine, I'll track the time it takes for the simulations to run. By comparing these times against the time taken by the original software, I'll be able to measure performance.

The main obstacle I'll face during this project is time. As this is a new approach, optimizing certain parts of code may take longer than expected, leaving me running behind schedule. This shouldn't be a problem though, as I have a contingency plan (described in the Timeline section) that should allow me to stay on track despite any setbacks. Other than that, I don't expect too many issues, as *∂a∂*i is fairly stable at this point, and thus shouldn't present any surprises.

Timeline

Before April 22 (Acceptance Announcements)

- Read about population-simulation software, the type of data it generates, and its applications. This will allow me to understand the data generated by the software.
- Install *∂a∂*i software onto my computer and all necessary dependencies. Work through the manual and the provided examples to make sure I understand how to use the software and what it does.
- Study the new approach for solving differential equations, to understand how the mathematics work.

April 22 - May 22 (Community Bonding)

- Talk with my mentor and possibly Dr. Gutenkunst to further discuss expectations for the summer project. This is an area of active research, so it is critical that I'm aware of any developments and how they affect the project plan. Get to know the community involved in the project and understand my role in it.
- Look through the source code for the prototype of the computational engine. Begin to think more specifically about how I might optimize the engine. Make sure I understand how exactly the engine implements the mathematical algorithm, and what each function does.
- Look at the source code for the original project to see how my new implementation will have to be structured in order to properly interface with the existing code-base.
- Decide upon some sample data sets, possibly the example data sets provided with the software, that I can use to have a consistent way to measure performance of the new computational engine and compare it to the previous software.

May 23 - June 5 (Coding Period Begins)

- Profile the code to take an initial note of which functions take the most time. Also take initial measurements of the time it takes the software to solve the sample data sets.
- Port the computational engine to C++ to improve performance.
- Embed the new C++ code in the existing code-base, and run tests to ensure that it correctly interacts with the software.

June 6 - June 19

- Optimize the matrix generation algorithm. Provide an option for the user to store their generated matrices in memory, to save time when running subsequent analyses.
- Optimize the algorithm for resolution of the linear system (Ax = B). Take advantage of approximating A to be a pentadiagonal matrix, as these matrices are especially convenient with numerical analyses.

June 20 - June 27 (Midterm Evaluations)

- Week of buffer time to finish up any unforseen issues with the previous steps.
- Provide test results to show that code acts as expected.
- Time the new engine on the sample data sets. Provide comparisons between these times, the times at the beginning of the project, and the original engine to show improvements made so far.
- Submit my midterm evaluation.

June 28 - August 5

- Profile the code again, to see if any of the minor functions that are called frequently are taking up too much time.
- Work to optimize the algorithms for any time-wasting functions found.

August 6 - August 14

- Work on the optimization algorithm for comparing our simulations with data. This involves finding a better way to determine the parameters that correctly fit the data.
- If necessary, this will be used as another period of buffer time, to finish up previous steps due to unforseen issues. See the contingency plan below.

August 15 - August 23 (Final Submission)

- Clean code and add documentation to make it easier to understand.
- New documentation will be mostly unnecessary, as ∂a∂i is already well documented and nothing has changed from the user's side, because this project worked on internals. However, the documentation can be updated to reflect the new computational engine, how it works, and the advantages it provides (i.e. users can now analyze more complex datasets). Furthermore, it is possible that additional options will be provided to the user after completion of this project (e.g. choice to store matrices in memory), which will need to be documented as well.
- Run final tests on the new engine. Also perform a final timing on the sample data sets, to find the overall performance enhancements since the beginning of the summer.
- Final computational engine will be located in C3G's independent ∂a∂i package, and possibly be pushed to the original package created by Ryan Gutenkunst as well (not determined yet).
- Submit final code sample as determined by discussion with my mentor, Dr. Gravel.

August 23 - August 29 (Final Evaluations)

- Submit final evaluation.

August 30 (Results)

- Celebrate!

CONTINGENCY PLAN

I've provided some buffer time throughout the project to allow me to catch up in case I run into any issues. Additionally, in case a major unforseen difficulty comes up, I will skip improving the optimization algorithm used to compare simulations to data. This will allow me to place my focus on the main issue: optimized solution of the differential equations. This plan should be more than enough to allow me to complete the main project no matter what issues I run into.

Management of Coding Project

Code will be committed as each major step is completed (i.e. Python ported to C++, new matrix generation algorithm, new linear system resolution solver, rewrite of remaining minor functions, etc.). This will probably mean committing around once a week, but I will be in constant contact with my mentor to keep him updated on my progress and ensure that I am still on track. If I fall behind or run into an issue, I will also be sure to discuss this with my mentor to get advice on how I can solve the problem, and how I can get back on track.

Testing will occur throughout the summer, as described in previous sections. Additionally, I will work to make sure that my code is clean and well documented as I create it, so that there is less work to do at the end of the summer. Updating the computational engine is an incremental process, so I will test everything regularly to detect problems as soon as they occur.

Test

My qualification test was to complete an IPython notebook (located here) that dealt with the evolution of allele frequencies in a finite population. Completion of this notebook involved:

- Learning the basics of genetic simulation programs. The program in the notebook implemented a Wright-Fisher simulation model for studying the evolution of allele frequencies in a simple population. I learned about the parameters necessary for definining populations, various ways one can choose to model evolution, and different ways we can represent this data in Python.
- Implementing the model using Python, including packages that will be important during this project (e.g. numpy, mathplotlib, IPython). Most of the code necessary to provide structure, plot graphs, etc. was provided, but I had to add the lines critical to actual implementation of the model.
- Computing the mathematics behind the model, by applying probability theory to determine the type of distribution expected in different scenarios (e.g. hypergeometric, binomial, Poisson), or calculating the expected value for allele frequencies, fixation, etc. I solved these problems theoretically, and then proved that the empirical results from the code verified my calculations.