Parameter Fitting with PSO

New Mexico
Supercomputing Challenge
Final Report
February 25, 2017
LAHS-4
Los Alamos High School

Team Members
Alex Ionkov
Phillip Martin

Teacher
Adam Drew

Project Mentors
Latchesar Ionkov
Bill Hlavacek

EXECUTIVE SUMMARY

Models are developed in order to simulate problems. They are considered useful for understanding complex processes and phenomena. However models have no basis if they are not fit to experimental data. Fitting algorithms can be used to search the parameter space of the model to make it match the experimental results. In this project we focus on biological modeling. BioNetGen Language (BNGL) is a programming language specifically designed for development of chemical kinetic models of cell signaling systems. BioNetFit is a fitting application made to automate the process of fitting a BNGL model to experimental data. Managing complex models in terms of computation time and speed is problematic for many reasons. There is no support for CPU and GPU scaling which results in slow runtime. The most commonly used solution is to lower the volume at which the simulation is run however this may cut out certain components of the system. In this project, we offer a better solution. Often when simulating large models, specifically in biochemical systems, the sample size or volume is lowered. The issue with this is that, for example, in a cell there may be five copies of a protein *A*, and when a tenth of the cell is simulated, protein *A* will not interact with any other proteins because it has been cut out of the model through downsizing. Therefore, a balance must be found between running the entire volume of a cell and one small fraction. We chose to write a function that would enable volume scaling while the chosen algorithm is fitting the model. The program checks if the parameters are "fit" every time a process completes, if they are "fit" BNF scales the simulation size up and this continues until the maximum volume is met (ie 1).

INTRODUCTION

Models are developed in order to simulate problems. They are considered useful for understanding complex processes and phenomena. However, models have no basis if they are not fit to experimental data. Applications are made to automate the process of fitting models to experimental data.

In this project we focus on biological modeling. BioNetGen Language (BNGL) is a programming language specifically designed for development of chemical kinetic models of cell signaling systems. BioNetFit is a fitting application made to automate the process of fitting a BNGL model to experimental data. Managing complex models in terms of computation time and speed is problematic for many reasons. There is no support for CPU and GPU scaling which results in slow runtime. The most commonly used solution is to lower the volume at which the simulation is run. However, this may cut out certain components of the system. In this project, we offer a better solution.

DESCRIPTION

Problem Statement

Often when simulating large models, specifically in cellular signaling, the sample size is lowered. Below is an code excerpt from a model.

```
begin model

begin parameters

# fraction of a cell to consider in a simulation

f 1

# Avogadro constant

NA 6.02214e23 # molecules per mol
```

The fraction *f* of the cell is defined as “1”. This means the model is simulating an entire cell. For complex models, simulating an entire cell is very computationally expensive. Many scientists simulate at lower fractions of a cell (i.e. 0.1) in order to receive data faster. The issue with this is

that, for example, in a cell there may be five copies of a protein *A*, and when a tenth of the cell is simulated, protein *A* will not interact with any other proteins because it has been cut out of the model through downsizing. Therefore, a balance must be found between running the entire volume of a cell and one small fraction.

Considered Methods

Models are made to simulate lower population sizes to save time however the results are sometimes skewed because of this. This problem is widely recognized and researched. Many methods have been defined, but few have been implemented. Those considered include rewriting the simulator to support GPUs, adding support for scaling across multiple CPUs (and/or GPUs), or editing BioNetFit to allow model scaling during the fitting process.

The simulator (NFsim) currently does not support GPU usage or scaling. Parallelizing, especially for GPUs, would require rewriting both NFsim and BioNetFit, resulting in entirely new simulators. Due to time constraints and complexity, this is not a viable option.

Thus, we chose to write a function that would enable volume scaling while the chosen algorithm is fitting the model. The program checks if the parameters are "fit" every time a process completes, if they are "fit" BNF scales the simulation size up and this continues until the maximum volume is met (ie 1).

BioNetGen Structure

BNGL is a rule-based programming language which entails specification of necessary and sufficient conditions for a reaction to occur, as well as parameters governing the rates of the reactions.

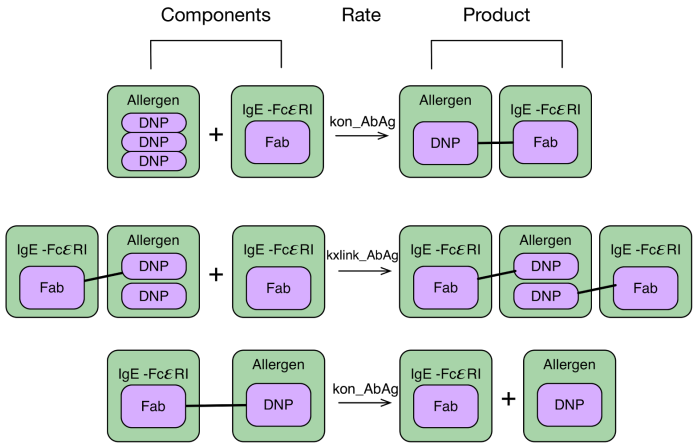

Figure 1: Visual representation of BioNetGen rules.

BioNetFit Structure

BioNetFit (BNF) offers many algorithms for fitting: genetic algorithm, differential evolution, and particle swarm optimization. An analysis of the algorithms offered is shown in Figure 2.

Algorithms		
Genetic Algorithm	Particle Swarm	Differential Evolution
define population		
in each iter.: generation	a particle is a parameter	each parameter is an agent
after a number of generations, best fit is decided	particles move towards a "best fit"	agents move, if the new position is "better" it is kept

Figure 2: Table comparing different BioNetFit algorithms.

Depending on the chosen algorithm, the user can choose the amount of generations and permutations. There are a certain number of permutations in each generation. BNF then parses the the model file and searches for "free" variables with the definition of

```
variablename__FREE__
```

Next, the algorithm that was chosen runs for the amount of generations and permutations. After every generation, BNF picks the best fit parameters and starts the next generation until it reaches Generation *X* where *X* is the amount of generations defined. Once Generation *X* is completed BNF provides you with the model file with the best fit parameters.

Code Structure for BioNetFit2

To run said algorithms, BioNetFit2 employs eight key C++ program files:

Config.cpp

BioNetFit requires a configuration file to be made specifying various parameters including the algorithm. The Config.cpp file parses this configuration file.

Data.cpp

Data.cpp provides functionality for manipulating data. It has functions that you can call to perform basic functions on the data.

FreeParam.cpp

Models have free parameters that are “free” or have no default value. By fitting the model, BioNetFit determines the parameters that give the best fit. FreeParam.cpp defines free parameters and how they are handled.

Model.cpp

BioNetFit parses the model file and replaces free parameters with generated ones. Model.cpp does the parsing and replacing as well as read the model for the volume for our function.

Particle.cpp

Particle.cpp defines the four types of algorithms used by BioNetFit. It defines particles, the particle swarm optimization, genetic, differential evolution, and simulated annealing algorithms, and calculates fitting. It also produces smoothing permutations to smooth the output.

Pheromones.cpp

Pheromones.cpp is specific to particle swarm optimization. It defines the attraction of particles to the “best” or center particle.

Swarm.cpp

Swarm.cpp defines variables for the swarm in particle swarm optimization and mutations in all algorithms. It also calculates the parameters for the following permutations.

Utils.cpp

Utils.cpp provides functionality for file i/o more specifically writing and reading the model files.

The preceding analysis was done because in order to write a function that works with the original code, one must understand the original code.

RESULTS

We modified the Particle class and added our function. Our function scales the volume up by an interval the user specifies after the simulation is determined to be reasonably fit at the current volume. During our tests we uncovered multiple bugs unrelated to our code and submitted fixes to BioNetFit2 maintainers. Unfortunately, we could not complete our tests because of bugs that have not been fixed yet. The major bug found was BioNetFit2 crashing after the first generation of simulations, which wouldn’t allow our function to occur.

CONCLUSIONS

We worked on improving the lives of biologists by reducing the time it takes to fit their models to experimental results. We learned about various fitting algorithms and their advantages and disadvantages, primarily focusing on particle swarm optimization. In addition, we contributed to an open source project that is in active development by multiple academic institutions. We were

unable to get concrete results to test the effectiveness of our added functionality because of bugs in the BioNetFit code that we were unable to patch in the short amount of time we had to work on our project.

RECOMMENDATIONS

First and foremost, we must fix all the bugs in the BioNetFit code that prevented us from testing our function. As stated before, our method improves the parameter fitting workflow by reducing the time it takes to complete. However, a more complete fix would be parallelizing NFsim and BioNetFit. The benefits of parallelization, especially on GPUs, would quite literally blow our function out of the figurative water. We could not accomplish this this year because of time and the steep learning curve of parallelization specifically CUDA.

ACKNOWLEDGEMENTS

We would like to thank Dr. Bill Hlavacek, and Dr. Ryan Suderman for help and inspiration for this project. We would also like to thank Latchesar Ionkov for help with the many technical difficulties that befell us.

REFERENCES

A.M. Smith, W. Xu, Y. Sun, J.R. Faeder, G.E. Marai, RuleBender: Integrated Modeling, Simulation and Visualization for Rule-Based Intracellular Biochemistry, *BMC Journal of Bioinformatics*, **13**, (2012), 1-16.

Chylek, L. A., Harris, L. A., Tung, C.-S., Faeder, J. R., Lopez, C. F. and Hlavacek, W. S., Rule-based modeling: a computational approach for studying biomolecular site dynamics in cell signaling systems. *WIREs Syst Biol Med*, **6,** (2014), 13–36.

Chylek LA, Holowka DA, Baird BA and Hlavacek WS, An interaction library for the FcεRI signaling network. *Front. Immunol*. **5**, (2014), 172.

J. R. Faeder, M. L. Blinov, and W. S. Hlavacek, Rule-based modeling of biochemical systems with BioNetGen. *Methods Mol. Biol.,* **500**, (2009), 113–67.

Kaur, Jaspreet, et al. "Parallel Implementation of PSO Algorithm Using GPGPU." 2016 Second International Conference on Computational Intelligence & Communication Technology (CICT), 2016, doi:10.1109/cict.2016.38.

M. W. Sneddon, J. R. Faeder, and T. Emonet, Efficient modeling, simulation and coarse-graining

of biological complexity with NFsim. *Nat. Methods,* **8**, (2011), 177-183.

Nedjah, Nadia, et al. “Parallel Implementations of the Cooperative Particle Swarm Optimization on Many-Core and Multi-Core Architectures.” International Journal of Parallel Programming, vol. 44, no. 6, May 2015, pp. 1173–1199., doi:10.1007/s10766-015-0368-3.

Tan, Ying, and Ke Ding. “Gpu-Based Parallel Implementation of Swarm Intelligence Algorithms.” Transactions on Cybernetics, vol. 46, no. 9, Sept. 2016, pp. 2028–2041., doi:10.1016/c2015-0-02468-6.

Ting, Tiew On, et al. “Multicores and GPU Utilization in Parallel Swarm Algorithm for Parameter Estimation of Photovoltaic Cell Model.” Applied Soft Computing, vol. 40, 2016, pp. 58–63., doi:10.1016/j.asoc.2015.10.054.