KNN UMAP Project learnings and Overview
Lead by: Tyler Biggs
Team: Mitchell Greer
Report by: Mitchell Greer

Goal:

The goal of this project was to create topographical maps of cooperating gene structures during a specific treatment. This was to be done by taking a GEM and using the UMAP software to generate a topology for the matrix, then comparing these topologies for different GEMS to get a sense of what genes were working together in what situations.

Technologies:

GEMMaker:

We use GEMMaker in this project to convert RNAseq data into GEMS to process into the topographical map of gene structures.

KNN Analytic:

This is an analytic created using the ACE libraries to calculate the full KNN for the GEM provided. The goal of this program is to calculate the full KNN of each gene in the GEM. It uses the city block distance metric to calculate the KNN and is very greedy when it comes to resources. For a reference of runtime, it should take a file that has 55,000 genes and 475 samples and calculate the full KNN within 6 hours. There are some things to note when using this tool:

1. The file path for the OpenCl program that it uses in the file “knn_dist_calc_opencl.cpp” uses the absolute path for the file. This will cause problems when you are trying to use this program as it is unlikely my absolute path is the same as anyone else's. To fix any problems with the file path, simply change it to the absolute path of the OpenCl file it uses, called “dist.cl” in the opencl folder within the src folder in the github repo.
2. While using this program (locally), if you are running it on your GPU it will make the computer virtually unusable, this is because it will grab all of the resources it can from the GPU and the CPU to process the full KNN faster.
3. There is no CUDA implementation.

UMAP:

We utilize a python library called UMAP to lower the dimensions of the GEM and create the topographical structure of the cooperating genes. This requires the KNN list that is created by the KNN analytic, as well as the GEM. You don’t have to supply the KNN values themselves, if you are just using a small number of K, however if you want to do a full KNN it is suggested to use the analytic to crank out the full KNN list before hand. There are a couple of things to note about some aspects of using this python module:

1. The size of input KNN indices and KNN distances can be quite large, for instance for the dataset mentioned above of 55,000 genes and 475 samples, the corresponding files are 17 gigs and 24 gigs respectively.

2. The method we utilize creates the structure and corresponding weights for each point in the structure, currently we don’t have a fast way to graph this, but we have been using a forced atlas2 layout to graph it in a readable way.

Project Status:

At the time this report was written The project was near completion. There are two main problems that are being faced, and upon resolution will yield a complete project. Those problems are:

1. Importing the KNN data into python in such a way that you don’t need to utilize ram. This is discussed more in the “Size of Data” roadblock.

2. Graphing the created topography. There have been two main approaches for doing this. The first is using a python module containing a method to shift the data using the ForcedAtlas2 algorithm, however this approach proved to be a little slow. The second is to use another tool built by the SystemsGenetics group called BioDeepVis. THis tool uses the GPU to run the ForcedAtlas2 algorithm much faster, however it does not account for weights.

The steps to solve these problems are not specified here, as they are not established. Either approach to graph the information could be used, though it would be faster in completing to use the first one, however this may prove to be a lengthy weight time for visualization. The first issue is more open ended, and it is not known whether using dask arrays will even work with our data.

Roadblocks Encountered:

Size of Data:

The shear size of the data we are expecting the user to have when using this tool is large, this led to problems manipulating the dataset. This was not as much of a problem with the KNN analytic, but became apparent when trying to use the results of the analytic in the UMAP function. We looked into using numpy memory maps to store the data on disk in a binary fashion, but came to the conclusion that this route could not be the solution, since the max size of a memmap is 2 gigs, and we needed to map 41 gigs of data. We have looked into using dask arrays to complete this task, but did not get to implementing this.

Competing Solutions:

Over the last few months another group has created a python module that allows the user to do this work at a much faster pace, this module is called “cuML” and was developed by a group named RAPIDS. It is unclear if this module will be able to complete the goals set forth at the beginning of the project, but if the module is ever actually importable, it looks very promising in that respect.