Data Storage Report
Jake Mikouchi

For this research project, I was tasked with creating a code that can effectively read a series of txt files, and condense the important information into a single file. The code would have to read thousands of txt files and store the relevant information, because of the amount of data being stored, data serialization would be needed to efficiently store the needed information.

Data serialization is the process of storing complex data by converting it into a stream of bytes. It is primarily used as a way to condense large amounts of data for storage and distribution. There are many different methods of data serialization each having their own pros and cons. Going into this project, my first task was to find the method of data serialization which would work best given the data we were storing and how the data was going to be used after storage.

I looked into 6 different forms of data serialization. Among these 6, there were 3 that are incompatible with the data we are storing.

1. Json
    - Pros:
        being available in almost every programming language
        most lightweight of any of the methods.
    - Cons:
        in the amount of data that can be stored
        stores the data in a binary format
2. Bson
    - Pros:
        can store more types of data than Json
    - Cons:
        many of the same cons as Json
        often takes up more space than Json.
3. Yaml
    - Pros:
        with using text editors
        is able to reference data within
    - Cons:
        same cons as Json and Bson
4. Python Pickle
    - Pros:
        highly customizable and easy to work with within python
        does not lose precision
        able to be compressed further using zip

- Cons:
  has a size limit
  Only available in python

5. Parquet
   - Pros:
     precise with up to 15 digits
     available in many languages
     can be compressed using zip
   - Cons:
     Not human readable

6. Hdf5
   - Pros:
     N-dimensional storage
     availability in almost every programming language
     no limit to the size of the file
   - Cons:
     Not optimized for python

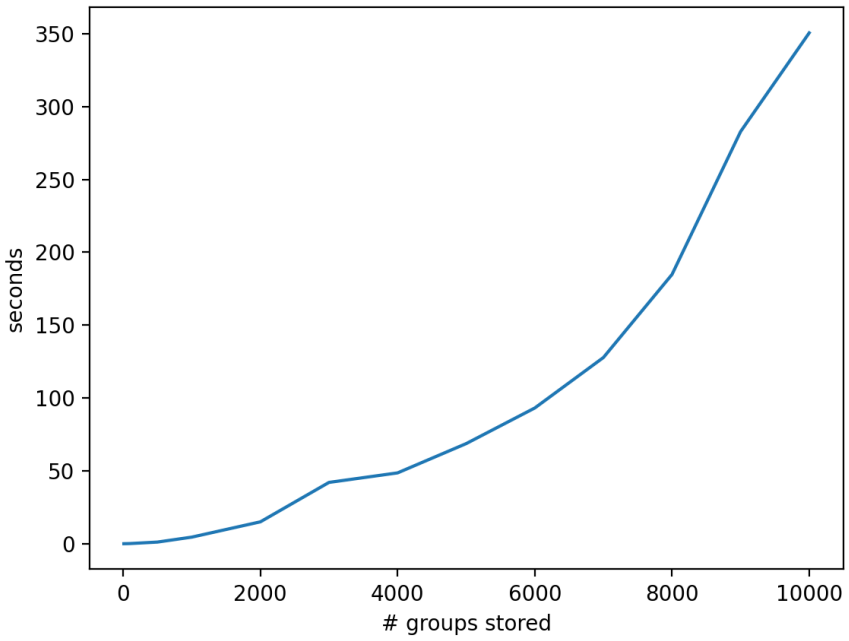
After some consideration, we decided to use Hdf5 largely for its use of multi-dimensional storage. Hdf5 does have a learning curve but with some practice it becomes much easier to navigate and store data within. Using Hdf5 I am able to store data in intuitive ways exactly in the format I wish to store it.

The python scripts are run using an anaconda build of python, which allows us to edit Hdf5 files. There are three main scripts, Construct_Sim3_hdf5.py, InpOutRead.py, and CleanHdf5.py.

- Construct_Sim3_hdf5.py:
  This script is over 400 lines long and does everything required to construct the Hdf5 file including
  - Creating a general information group
  - Storing a general Fuel Assembly layout as well as general Core Conditions
  - Storing the incoming data including the fuel assembly, core conditions, and objective goals
  - Comparing the Fuel Assemblies and objective goals to the information in the general information group so that the conditions for the optimization problems stay consistent

- InpOutRead.py
    - This script stores the path to the input and output files given by the user
    - The purpose of having a separate script for this is user convenience, so that the user does not have to open and locate information within a large script every time the path changes

- CleanHdf5.py
    - This script runs through the hdf5 files created by the other scripts and deletes repeat information
    - This script works in the most time efficient manner but as the number of stored groups within the hdf5 increase, this script may not be able to run, however the script was able to run with 100000 groups in testing and may be capable of much higher.
      In case the script is not able to run, there is a CleanHdf5backup.py script that is able to work with more information but is much slower
    - The main drawback of this script is that as the amount of groups gets larger the amount of time it takes to run increases exponentially as shown here
    - 100000 groups took approximately 2 hours to complete



Together the scripts are able to work fast and efficiently, adding a new set of data takes anywhere from 0.01 to 0.05 seconds even when there is a large number of data sets already stored. The CleanHdf5 script is more complex and runs at a Big-O complexity of O(n^2) to a moderate degree. As more data is in the script that will be cleaned, the time to run the script increases at a quadratic rate. However, it should be noted that the rate of increase starts quite slowly. Cleaning through 100 files takes about 0.12 seconds, cleaning 1000 takes 4.5 seconds, cleaning through 5000 takes 68 seconds and cleaning through 10000 takes 350 seconds. The tests that the cleaning script went through were unrealistic and in a real world scenario may be much faster than the numbers given here. The useability of the scripts is very easy, the only requirement from the user is that they give a path to the Input and Output between each run of the main script. Once the amount of data stored into the Hdf5 file gets very large, it is recommended that the user runs CleanHdf5.py or CleanHdf5Backup.py overnight / over the weekend, so that the program will not disrupt workflow.