Alex Markules, Grant Farnsworth, Jacob Kampf

Scheduling Spikes Write up

Senior Design I

12/12/18

Small Scale Tests:

For our first round of testing, each of us implemented a brute force scheduling algorithm in a different language. We tested these programs using a subset of the cohorts we were given, consisting of 5 cohorts, which each needed to be assigned to 3 to 5 classes. The requirements here can be found in the file called cohortReqs.csv, and the list of classes used can be found in the file classList.csv.

C:

One implementation was written in C, it iterated through each schedule recursively and generated schedules one at a time. This scheduler performed well in our tests, creating all possibilities in 7 minutes and 50 seconds. One thing we learned from this was that the number of possible combinations is exceptionally large, creating about 6 million possible schedules. Due to the vast number (and space requirement thereof), we implemented a second version of this algorithm, which more closely resembled a branch and bound method. This version tested each new class added to a schedule for conflicts in class time and class capacity. This version of the scheduler generated a more reasonable 652,080 schedules, in as little as 44 seconds.

C++:

Another implementation, written in C++, performed well in the first round of testing, though still slower than the C algorithm. It utilizes a brute force permutation algorithm, which with large input becomes an issue. This version generated over 5,529,600 schedules in about 6 minutes.

Java:

This brute force implementation uses a recursive algorithm called Heap's algorithm in order to find every possible permutation of the required classes for each cohort. Even on a small scale, the program took longer than the C and C++ implementations to run, which showed it would not be a viable option as we went forward.

Conclusion:

Since the Java scheduler performed much slower than the other two, we opted to remove it from the next round of testing, as it would likely take an unreasonable amount of time to complete the full list of 25 schedules. Our first round of testing made us cautiously optimistic that scheduling could be done in a surprisingly quick amount of time. However, exponential growth in the number of schedules as the number of cohorts increase means we'd need to do a larger scale test to be confident in our scheduling.

Full Scale Tests:

For this test, we attempted to use the C and C++ algorithms from before to generate a full list of schedules. The course requirements used were for all 25 cohorts that the school plans to use for the Fall 2019 semester, and the classes used were all the Fall 2018 course offerings relevant to

those requirements (the exact files used are “cohortReqsLarge.csv” and “classListLarge.csv”). These tests required an exponentially higher amount of time compared to the first set.

C Brute Force:

While the C brute force algorithm performed well in the first round of testing, it proved to be much slower in the full scale test. The first issue was space. The output of this scheduler proved to be far larger than any machine we had access to could handle, generating 10 Gigabytes of output in the first few minutes, before generating even 1% of the total possibilities. Since our final product will rank schedules, and only store the ones that rank highest according to our criteria, we decided to run the program without print statements. This still performed poorly, running for less than an hour before crashing with no output.

C Branch and Bound:

This version of the program seems to be performing better than the brute force one. It again ran into the issue of generating more output than our server could handle, causing it to run out of RAM space and crash, but after again removing print statements and fixing any memory leaks, we found it to run in what seems to be a memory efficient manner, never exceeding more than 70 MB of space used. The program has been running on our test server for nearly 12 hours. We hope that this process will finish within a few days. If not, we may need to look into ways of limiting the branches which are explored further, or using a more effective algorithm to generate schedules.

C++ brute force:

Since this implementation enumerates all possible combinations as integer arrays, the increase in number of cohorts required a massive amount of space. The program crashed immediately due to lack of memory. It will require complete refactoring in order to work for the full list of cohorts.

Final conclusion/Next steps:

Since all brute force implementations failed to handle the full list of cohorts, we will have to wait for the branch and bound implementation to finish processing before coming to a conclusion about the validity of that algorithm. If it finishes within a few days, we will accept that as our ideal algorithm, and look into methods of tweaking it to improve speed, like eliminating branches when their score exceeds that of our top 20 current solutions, and spinning up multiple threads to process the various solutions. If the program runs for more than 100 hours, it will likely be insufficient, especially if the advisors ever decide they would like to add more cohorts than exist currently. In that case, we will need to look into more advanced methods of calculating schedules, such as stochastic computing.