Class Scheduler

Zhiyu An

May 2023

1 Introduction

This work presents an efficient algorithm for class scheduling which uses less computational requirements than $O(n!)$, where n denotes the number of timeslots. Our approach exploit the inherent structure of the problem, specifically the fact that each class is allocated to a limited number of timeslots, thereby shrinking the search space for each student.

2 Simplest Example

Suppose we have four classes AP Statistics, English 500, History, and PE, which needs to be scheduled to three timeslots. The availability is as follows:

slot 1	slot 2	slot 3
AP Statistics English 500	History PE	AP Statistics English 500

Based on the availability chart we construct a dictionary of classes where the values are the available timeslots for each class:

```
dict_class = {'AP Statistics': (1, 3),
              'History': (2),
              'PE': (2),
              'English_500': (1, 3),}
```

Suppose students Alice and Bob each chooses the following classes:

Alice	AP Statistics, PE, English 500
Bob	AP Statistics, PE, History

Then for each student, we compute the Cartesian product of the availability of their chosen classes, where the index of the products are kept the same as in the students' preference table. For Alice, the Cartesian product would be

```
(1, 2, 1, )
(1, 2, 3, )
(3, 2, 1, )
(3, 2, 3, )
```

If there exists sets of available timeslots in the above products where each element is unique (e.g. (1, 2, 3,) and (3, 2, 1,)), then those sets are valid schedules for the student. We can then return the class titles along with their timeslot number:

```
[('AP Statistics (slot 1)', 'PE (slot 2)', 'English 500 (slot 3)'),
('AP Statistics (slot 3)', 'PE (slot 2)', 'English 500 (slot 1)'),]
```

In this case, a valid schedule exists for Alice but not for Bob, as there is no scenario where the timeslots for bob's classes are unique. Note that for Alice, we only had to determine the uniqueness of 4 sets, instead of $3! = 6$ sets.

3 Time Complexity

In scheduling for n timeslots, a conventional permutation-based approach has a computational complexity of $O(n!)$ for each student. Alternatively, our approach's computational complexity is determined by the number of available timeslots for each class, i.e. how frequent each class is being offered.

We observed that each class has a small number of timeslots allocated for them in practice, usually only one or two timeslots. This is because a diverse set of classes are offered in a relatively small number of timeslots. Therefore, it is more efficient to iterate through available timeslots for classes than to iterate through all timeslots and then compare them with the classes' availability.

Assuming each class has an average of m timeslots, our method scales at $O(n^m)$ computations per student. Therefore, our approach exhibits superior efficiency when $m < (n!)^{1/n}$. For instance, when $n = 7$, our method requires less computation provided $m < 3.38$.