CS340 - Project: The Registrar's Problem

Amina Ahmed, Joon Luther, Foqia Shahid

November 14, 2022

1 Abstract

The goal of scheduling courses is to maximize the number of students who can enroll in their preferred courses in order to meet the college wide and major requirements to graduate. There are many types of conflict that can arise during scheduling that must be avoided in order to produce a valid schedule. Minimizing student schedule conflict helps ensure that a majority of students can enroll in their preferred courses.

In order to take student preferences into account when designing the schedule, the proposed Greedy algorithm prioritizes the course popularity among students. A course with many interested students will take precedence in the scheduling process over a course with fewer interested students. The benefit of this decision is that courses with the greatest number of interested students are more likely to be scheduled to minimize conflict.

After implementing such an algorithm, it can be run on randomly generated data with varying complexity to test the optimality of the algorithm. Additionally, inferences regarding certain constraints and considerations for potentially improving the optimality of the resulting scheduling can be made. increasing available time slots, increasing the number of classrooms, etc. are some of the recommendations that are explored in this paper with experimental analysis.

2 Algorithm Description

The input is given as a set of constraints, including student preferences, time slots, professors and classrooms. Our proposed algorithm prioritizes the popularity of courses in scheduling by generating a count for each class based on how many times it appears in student preferences.

The algorithm starts by considering each classroom in order of size, from largest to smallest. For each of the time slots and classroom combination, there exists a corresponding list of students who are available for class during that time slot. The list of available students is then traversed through to find the class that the most amount of students want to take. This can be labelled as the "most famous class" for the corresponding time slot.

This class is then assigned to the current time slot and room, given that the professor who teaches that class is available during that time slot and the room is eligible to hold that class (Eg. rooms in park are eligible to hold computer science courses). If any of these requirements can't be met the next most famous student is found. Once a class is found that can be taught in the classroom and for which the professor is available, the room and the time slot are assigned to the class.

Once the class is scheduled, the students who prefer to take this class are then removed from the list of available students for that time slot. After a time slot and room is assigned to a class, the algorithm moves on to the next time slot and room combination.

A separate function is responsible for enrolling each student into as many courses from their preference list as possible without exceeding room capacity or causing individual conflict.

3 Constraints Considered

3.1 More or less than 4 preferences per student

We assign either 3, 4 or 5 course preferences for a given data set.

3.2 Fewer than 2 professors per course

To implement this, we assign the number of teachers randomly between (number of classes / 2) and number of classes. Thus, one extreme is all professor teach two courses or the other extreme end is that each professor teaches only one course.

3.3 Overlapping time slots

We randomly generate overlapping time slots, allowing for only pairs of overlapping time slots and store the results in a hashmap. We can have atmost $\frac{T}{3}$ timeslots.

Then, in our assignment loop, we can run into two issues: professor conflict and room conflict. During assignment loop, we check that the professor of current most famous class is not already teaching in the time slot. In addition to this, we will now also check if the professor is teaching in the overlapping timeslot.

To check for room conflict, we made a new data structure that stores room to time slot assignments. Then for a given classroom, before assigning the most famous class to that room, if the timeslot has an overlapping timeslot, we check if the room has a overlapping timeslot assignment in it.

3.4 Scheduling in specific classrooms

Before scheduling a class in a room, the function isValidRoom() is used to check if that class can be scheduled in that specific room. isValidRoom() returns true if the room is belongs to the set of predetermined rooms that the class can be scheduled in. If the class can not be scheduled in the class room, then we consider the next class.

3.5 Scheduling sections of a course

A class is broken into two sections if the demonstrated interest is 1.2 times the average. This change is reflected in the preprocessPref() function which first calculates the average interest in a class. Then if a class is big enough, it is removed from the classCounts array and broken into sections. Those sections are added back to classCounts. This change is also reflected in the student-preferences.

This allows sections to be considered separately when scheduling them and is conflict adjusted.

4 Pseudocode

```
Function makeSchedule()
    for i:1 to classrooms.len do
        for j:1 to timeslot.len do
            if room is occupied in an overlaping timeslot then
                continue
            end
            if timeslot[j].availableStudents is empty then
                continue
            end
            if the professor is busy in this timeslot then
                prof = true
            end
            if the class can be scheduled in this room then
                class = true
            end
            if professor is teaching in any overlapping timeslots then
                overlap = true
            end
            while overlap or !class or prof do
                mfclass = most famous class timeslot[j].availableStudents
                idx = index of mfclass in classCounts
            end
            classCounts[idx].assignedtime = timeslot[j]
            classCounts[idx].assignedroom = classrooms[i]
            remove mfclass from availableStudents
        end
    end
    return classCounts
```

Function `enroll()`

```
for Each student s_i do
    for Each course preference c_j of s_i do
        Look up the time slot of course c_j;
        if s_i is not already enrolled at time then
            if The number of students already enrolled in c_j does not exceed the
              capacity of r then
                Look up the Classroom assignment r of c_j;
                add s_i to the list of students enrolled in c_j;
            end
        end
    end
end
```

5 Time Analysis

Data assumptions:

1. Number of professors, p, can be at most c.
2. Number of rooms, r, and number of time slots, t, are small constants.
3. l_i is the length of a student preference list s_i. l_i is a small constant.

5.1 Pre-processing

During the pre-processing step, the algorithm stores time slots in $O(t)$, class rooms in $O(r)$, courses in $O(r)$, students in $O(s)$, and professors in $O(p)$. Then the time complexity of this step is $O(t + r + s + p + c)$. Assumption (1) reduces this expression to $O(s + p + c)$. Assumption (2) further reduces the expression to $O(s + 2c)$. Then, overall time complexity of pre-processing is $O(s + c)$.

5.2 Make schedule

The two For Loops in makeSchedule() work together to iterate through every classroom and time slot. This gives them a time complexity of $O(r \times t)$

Checking if the availableStudents list is empty for a certain time slot is of $O(1)$ time complexity. Additionally, checking if a room is occupied in an overlapping time slot has a time complexity of $O(t)$

Determining if a professor is busy in the time slot has an $O(p)$ time complexity. Similarly determining if a class can be scheduled in a classroom has an $O(r)$ time complexity. Finally, determining if a professor is teaching in any overlapping time slots has an $O(t + p)$ time complexity

The While Loop runs until it finds a valid class. This can have an $O(c)$ time complexity at worst. Similarly finding the most famous class in available students will at worst have an $O(s)$ time complexity. Lastly, finding getting the index for the most famous class will have an $O(c)$ time complexity.

Removing the most famous class from available students will have an $O(s)$ time complexity.

This give the makeSchedule() function a total time complexity of $O((r \times t)(t + p + r + t + p + c^2 s + s)) = O((r \times t)(c^2 s)) = O((r \times t \times c^2 \times s))$

5.3 Enroll

During enroll, the algorithm iterates over each student in $O(s)$. Within this outer loop, the algorithm iterates over each course in the preference list of student s_i, which has length l_i. Looking up the time slot for a preferred course c_p of s_i is $O(c)$, since we must iterate through every course in the list of all courses. Then, we check each scheduled course to see if s_i appears in another course at the same time slot as c_p. Then, retrieving the number of enrolled students and the max capacity of c_p is

6 Data Structures

A 2D array *Students* is used to store student preferences with each index corresponding to a student that can have up to 4 preferences. Looking up a student's set of preferences given the index of the student is $O(1)$.

The final schedule is stored in the array of $Course$ objects called *Schedule*. Each $Course$ object has a variable storing course number, time interval, room, professor, and a list of enrolled students. Adding a $Course$ to *Schedule* is $O(1)$.

In order to check if a professor p teaching course c has a time conflict for a time slot t, the algorithm iterates through all scheduled courses to see if p is teaching a different course than c also in time slot t. This process is $O(c)$.

Then, the schedule function runs in $O(c^2)$ time.

In order to check if a student time conflict exists, for a student s and course c at time t, the algorithm goes through each scheduled course to see if the student is enrolled in another course at t. This process takes $O(c)$ time.

Checking if a course has reached maximum capacity given a course c is $O(1)$ because the maximum capacity is an attribute of c.

Enrolling a student in a course c is $O(1)$ because c contains an Array List of enrolled students.

Then, the enroll function runs in $O(sc)$ time.

7 Proof of Correctness

Proof of termination: The schedule function runs for $r \times t$ iterations, terminating after every course has been scheduled. Then the enroll function runs for s iterations, terminating after every student has been evaluated and enrolled in non conflicting courses.

Proof of Validity: To prove that the schedule produced is valid, we want to ensure the following:

1. No professor is scheduled to teach more than one course at the same time
2. No room is assigned to more than one course at the same time
3. All courses that can be scheduled are in the schedule
4. No course is scheduled more than once
5. No student is enrolled in more than one course at the same time

By construction, (3) is true. All rooms are filled for all the time slots and all time slots are filled for each classroom. We are ensuring this since our outer loop runs for each classroom and the inner loop runs for each time slot and the assignment is done for each combination of classroom and time slot.

For (4), since are maintaining a data structure that contains student to course preferences and removing the most popular class after assignment from this list, we ensure that that class is never considered again.

For (5), we ensure no student has a schedule conflict during the enrollment step. A student is considered for all of their preferred courses. However, if a student prefers a course c but is already enrolled in a different course at the same time conflict, they are not enrolled in c. Therefore, (5) is true.

(2) is ensured by construction since for each room, we put a course in a distinct time slot and only iterate to the next room once all distinct time slots are filled for that room. Thus for the next room, all the time slots are available.

(1) is ensured by checking if the professor teaching is teaching another course at the same time slot. If this is the case, the next most popular course is considered until there is a course with no professor conflict that can be added to the schedule.

8 Experimental Analysis

Our worst case time analysis shows that run time grows with $students * courses^2$, so for each type of data and each algorithm, we create a scatter plot and regression line of run time vs. sc^2.

Algorithm on Random Data (With base constraints):

no.	Classes	Students	Times	Rooms	Time(ms)	Best	Experimental	Optimality	sc^2
1	66	557	17	64	78.2	2228	2163.7	0.97114	2426292
2	64	1455	49	83	187	5820	5778.2	0.9928179	5959680
3	82	777	31	70	127.3	3108	3073.2	0.9888031	5224548
4	60	920	50	74	136.5	3680	3659.2	0.9943478	3312000
5	56	454	37	100	77.6	1816	1801.2	0.9918502	1423744
6	52	605	38	67	88.3	2420	2399.5	0.9915289	1635920
7	62	686	35	89	128	2744	2719.5	0.9910714	2636984
8	96	411	54	100	105.1	1644	1642.5	0.9990876	3787776
9	88	1062	11	76	127.7	4248	3960.7	0.93236816	8224128
10	66	485	37	77	81.6	1940	1927.8	0.99371135	2112660
11	84	1481	58	93	228	5924	5893	0.99476707	10449936
12	64	610	43	74	115.2	2440	2425.3	0.9939754	2498560
13	58	1344	58	72	183.6	5376	5376	1	4521216
14	98	900	51	67	162.9	3600	3587.9	0.99663883	8643600
15	98	1404	30	62	201.2	5616	5531.9	0.9850249	13484016
16	72	1391	42	86	182.9	5564	5516	0.9913731	7210944
17	66	777	18	76	109.2	3108	3010	0.9684685	3384612
18	36	1423	57	80	158.2	5692	5692	1	1844208
19	36	1409	60	98	182.2	5636	5636	1	1826064
20	78	472	13	64	70.5	1888	1817.6	0.9627119	2871648
21	42	493	29	74	80.8	1972	1941.7	0.9846349	869652
22	88	309	11	93	68.5	1236	1196.2	0.9677993	2392896
23	96	1500	55	97	224	6000	5968	0.9946667	13824000
24	92	739	41	83	129.4	2956	2939.1	0.99428284	6254896
25	56	809	23	81	129.8	3236	3159.6	0.97639066	2537024
26	46	600	32	63	84.2	2400	2365.3	0.9855417	1269600
27	76	1006	42	69	174.9	4024	3995.4	0.9928926	5810656
28	44	917	11	97	99.4	3668	3392.2	0.92480916	1775312
29	90	1164	15	78	138.5	4656	4454.2	0.9566581	9428400
30	62	454	38	87	80.3	1816	1803.2	0.9929515	1745176
31	92	921	23	67	130.6	3684	3616.1	0.981569	7795344
32	78	1327	56	77	188.3	5308	5279.2	0.99457425	8073468
33	100	743	14	92	112.2	2972	2860.7	0.96255046	7430000

Figure 1: Table 1

Table 1 shows the result of running the algorithm on random data with the base constraints only. The scatter plot of run time vs. sc^2 produces a regression line of $R = 0.703$ and $R^2 = 0.494$. (Full table in repository under RandomData.xlsx)

Algorithm on Bryn Mawr College Data (With base constraints):

Table 2 shows the result of running the algorithm on Bryn Mawr data with the base constraints only. The scatter plot of run time vs sc^2 produces a regression line of $R = 0.952$ and $R^2 = 0.907$.

Semester	Year	Classes	Professors	Students	Times	Rooms	Time(ms)	Best	Experimental	%Optimality	s*c^2
Fall	2000	231	164	1112	58	60	414	3559	3294	0.92554086	59337432
Fall	2001	222	167	1098	59	59	399	3574	3352	0.93788475	54113832
Fall	2002	239	159	1090	63	61	490	3579	3277	0.9156189	62261890
Fall	2003	241	151	1104	59	59	434	3580	3331	0.9304469	64121424
Fall	2004	265	163	1125	57	51	454	3720	3489	0.9379032	79003125
Fall	2005	255	156	1127	54	52	463	3686	3473	0.9422138	73283175
Fall	2006	269	169	1167	65	63	532	3798	3563	0.9381253	84445287
Fall	2007	283	169	1148	59	62	503	3864	3650	0.944617	91942172
Fall	2008	284	175	1213	65	63	561	3923	3586	0.91409636	97835728
Fall	2009	264	164	1352	65	67	592	4303	3874	0.9003021	94228992
Fall	2010	288	174	1475	68	68	648	4874	4296	0.88141155	122342400
Fall	2011	280	172	1600	76	64	682	5067	4529	0.8938228	125440000
Fall	2012	293	175	1659	79	70	708	5266	4657	0.88435245	142423491
Fall	2013	320	179	1644	78	69	814	5152	4578	0.88858694	168345600
Fall	2014	280	183	1635	74	67	682	4920	4408	0.89593494	128184000
Spring	2001	222	167	1098	59	59	399	3574	3352	0.93788475	54113832
Spring	2002	239	159	1090	63	61	490	3579	3277	0.9156189	62261890
Spring	2003	241	151	1104	59	59	434	3580	3331	0.9304469	64121424
Spring	2004	265	163	1125	57	51	454	3720	3489	0.9379032	79003125
Spring	2005	255	156	1127	54	52	463	3686	3473	0.9422138	73283175
Spring	2006	269	169	1167	65	63	532	3798	3563	0.9381253	84445287
Spring	2007	283	169	1148	59	62	503	3864	3650	0.944617	91942172
Spring	2008	284	175	1213	65	63	561	3923	3586	0.91409636	97835728
Spring	2009	264	164	1352	65	67	592	4303	3874	0.9003021	94228992
Spring	2010	288	174	1475	68	68	648	4874	4296	0.88141155	122342400
Spring	2011	280	172	1600	76	64	682	5067	4529	0.8938228	125440000
Spring	2012	293	175	1659	79	70	708	5266	4657	0.88435245	142423491
Spring	2013	320	179	1644	78	69	814	5152	4578	0.88858694	168345600
Spring	2014	280	183	1635	74	67	682	4920	4408	0.89593494	128184000
Spring	2015	231	167	1081	62	52	455	3299	3094	0.93785995	57683241

Figure 2: Table 2

Algorithm on Random Data (With additional constraints):

Table 3 shows the result of running the algorithm on random data with the additional constraints implemented. The scatterplot of runtime vs sc^2 produces a regression line $R = 0.681$ and $R^2 = 0.464$. (Full table in repository under RandomData.xlsx)

no.	Classes	Professors	Students	Times	Rooms	Time(ms)	Best	Experimental	Optimality	sc^2
1	96	87	1188	16	71	236	4752	4338.1	0.91289985	10948608
2	62	38	506	42	69	168.3	2024	1981.1	0.97880435	1945064
3	92	100	1022	22	75	216.7	4088	3851.1	0.9420499	8650208
4	92	71	1339	14	87	184.5	4017	3733.7	0.9294747	11333296
5	86	84	880	39	64	299.1	4400	4235.9	0.96270454	6508480
6	62	58	1221	59	57	245.5	3663	3637.5	0.9930385	4693524
7	50	41	381	39	65	137.1	1905	1851.1	0.97170603	952500
8	34	32	1047	35	86	148.2	4188	4164	0.9942693	1210332
9	50	43	586	43	69	149.7	1758	1741.2	0.99044365	1465000
10	64	42	515	46	74	235	2060	2019.7	0.98043686	2109440
11	100	94	1428	58	52	438.4	4284	4244.8	0.9908496	14280000
12	60	62	791	33	57	227.1	3955	3791.2	0.9585841	2847600
13	98	96	1286	14	57	235.9	5144	4623.7	0.89885306	12350744
14	70	52	1317	30	59	287.6	6585	6257.9	0.9503265	6453300
15	32	19	1225	35	65	175.7	4900	4886.3	0.99720407	1254400
16	42	42	1008	23	87	175.4	4032	3875.5	0.9611855	1778112
17	90	93	894	26	56	210.4	3576	3396.5	0.94980425	7241400
18	50	51	1190	38	77	210.1	4760	4666.8	0.9804201	2975000
19	52	31	910	30	66	171.6	3640	3524	0.96813184	2460640
20	74	68	421	35	77	128.2	1263	1233.9	0.97695965	2305396
21	36	20	427	33	70	112.5	1708	1676.7	0.98167443	553392
22	58	56	1389	34	68	231.9	5556	5408.8	0.9735061	4672596
23	44	25	306	54	97	121.9	1530	1530	1	592416
24	64	70	633	37	64	149.1	1899	1865.6	0.9824118	2592768
25	74	54	558	25	68	142	1674	1619.2	0.967264	3055608
26	90	105	661	14	82	137.5	3305	2853	0.8632375	5354100
27	72	52	917	56	85	289	4585	4509	0.9834242	4753728
28	58	65	1154	45	83	272.3	5770	5627.4	0.97528595	3882056
29	62	56	1308	39	93	302.8	5232	5110.7	0.97681576	5027952
30	94	54	579	19	99	165.8	2316	2141.4	0.92461133	5116044
31	56	56	404	50	57	140.5	1212	1200.2	0.990264	1266944
32	98	99	788	41	51	271.5	3152	3066.1	0.9727475	7567952
33	56	36	377	52	69	200	1885	1866.7	0.9902918	1182272

Figure 3: Table 3

Algorithm on Bryn Mawr College Data (With additional constraints):

Semester	Year	Classes	Professors	Students	Times	Rooms	Time(ms)	Best	Experimental	% Optimality	s*c^2
Fall	2000	231	164	1112	58	60	481	3559	1225	0.3441978	59337432
Fall	2001	222	167	1098	59	59	523	3574	1031	0.2884723	54113832
Fall	2002	239	159	1090	63	61	563	3579	1012	0.28276056	62261890
Fall	2003	241	151	1104	59	59	560	3580	1197	0.33435753	64121424
Fall	2004	265	163	1125	57	51	582	3720	1186	0.3188172	79003125
Fall	2005	255	156	1127	54	52	583	3686	1980	0.53716767	73283175
Fall	2006	269	169	1167	65	63	722	3798	999	0.26303318	84445287
Fall	2007	283	169	1148	59	62	722	3864	1400	0.36231884	91942172
Fall	2008	284	175	1213	65	63	746	3923	803	0.20469029	97835728
Fall	2009	264	164	1352	65	67	837	4303	1010	0.23471996	94228992
Fall	2010	288	174	1475	68	68	1008	4874	1256	0.2576939	122342400
Fall	2011	280	172	1600	76	64	1138	5067	1268	0.2502467	125440000
Fall	2012	293	175	1659	79	70	1234	5266	1309	0.24857576	142423491
Fall	2013	320	179	1644	78	69	1259	5152	879	0.17061335	168345600
Fall	2014	280	183	1635	74	67	1157	4920	1141	0.23191057	128184000
Spring	2001	222	167	1098	59	59	523	3574	1031	0.2884723	54113832
Spring	2002	239	159	1090	63	61	563	3579	1012	0.28276056	62261890
Spring	2003	241	151	1104	59	59	560	3580	1197	0.33435753	64121424
Spring	2004	265	163	1125	57	51	582	3720	1186	0.3188172	79003125
Spring	2005	255	156	1127	54	52	583	3686	1980	0.53716767	73283175
Spring	2006	269	169	1167	65	63	722	3798	999	0.26303318	84445287
Spring	2007	283	169	1148	59	62	722	3864	1400	0.36231884	91942172
Spring	2008	284	175	1213	65	63	746	3923	803	0.20469029	97835728
Spring	2009	264	164	1352	65	67	837	4303	1010	0.23471996	94228992
Spring	2010	288	174	1475	68	68	1008	4874	1256	0.2576939	122342400
Spring	2011	280	172	1600	76	64	1138	5067	1268	0.2502467	125440000
Spring	2012	293	175	1659	79	70	1234	5266	1309	0.24857576	142423491
Spring	2013	320	179	1644	78	69	1259	5152	879	0.17061335	168345600
Spring	2014	280	183	1635	74	67	1157	4920	1141	0.23191057	128184000
Spring	2015	231	167	1081	62	52	508	3299	941	0.28523794	57683241

Figure 4: Table 4

9 Discussion of Algorithmic Choices

Our algorithm can be categorized as a Greedy Algorithm because a schedule is generated by its calculated popularity for a specific time slot. As long as there is no conflict, the next most popular course must be scheduled in the current time slot and classroom and cannot be rescheduled afterward.

When designing this algorithm, our main approach was to solve some of the conflicts by construction, which we achieved for classrooms and time slots. The natural scheduling and enrollment complications that arose were how to ensure professors and students did not have time conflicts, and that courses were not enrolled past their capacity.

To avoid scheduling a professor to teach more than one course in the same time slot, we simply keep track of which professors teach in each time slot as the schedule is being created. With the professors, it was an easy problem of solving since it just involved keeping track of which professors were teaching in a given time slot and making sure we don't assign the same professor twice. However, things got more complicated with the students. To solve the conflict problem for students, we decided to keep track of the students that are assigned in each time slot and removing those students who are taking classes in that time slot as we go. Thus, in our consideration of the next most famous course, we would only consider the available students. The algorithm was hard to design since there were many of variables to keep track of and special cases like when the same professor is teaching the top most famous classes.

10 Solution Quality Analysis

When testing both our constrained and unconstrained algorithms using random data, we achieved a high optimality in the schedules the algorithm output. This optimality persisted as the random data varied. We achieved a upper bound of 100 percent for both algorithms and a lower bound of 90 percent for the unconstrained algorithm and 85 percent for the constrained algorithm

We further tested our algorithm by randomly increasing or decreasing the number of classrooms, professor, time slots, courses and classrooms input into the algorithm but the optimality seemed to have a positive co-relation with each variable most of the time, especially when it came to time slots.

We believe the higher optimality is due of the even spread of the random data. Randomized data does not reflect and real life trends or interests therefore, it doesn't have much complexity that our algorithm had to deal with.

When we tested both algorithms with Bryn Mawr data, the optimality of unconstrained algorithm did not vary much and had an upper bound of 94 percent and a lower bound of 88 percent. The correlation, however, became negative with each input variable in both algorithms. This may be because with as number of professors, classes, time slots, class rooms and students increases, the complexity of the data provided also increases.

We believe the unconstrained algorithm faired better with the Bryn Mawr data because it was not programmed to handle the real life complexities of that data. By ignoring these, it produced a valid schedule, but not one that may fit very well into the real world.

Finally, testing the Bryn Mawr data with constraints gave an upper bound of 50 percent and a lower bound of 17 percent. This is believed to be because real life data is more complex than randomized data in the sense that it reflects trends and student interests and our algorithm does not handle such constraints. Recognizing the general interests of students in our algorithm may have given us a higher optimality.

11 Recommendations to the Registrar

1. When scheduling courses, our Algorithm always gives precedence to classes with the most interest also know as the most famous class. Similarly, there is a certain set of courses that have a high student interest each semester such as introductory courses, major requirement courses, general requirement courses etc.

   A list of these courses can be easily found by iterating over the data of past semesters. The interest in these courses will not vary significantly each semester and will be strong. Such classes can be schedule collaboratively between departments to minimize student conflict before student preferences are even considered. The remaining classes for the semester can then be scheduled around these classes after considering student preferences.

2. Each department has a set of requirements that a room has to satisfy to be eligible to hold a class for that department. Additionally, certain departments only consider rooms in certain buildings e.g Mathematics classes are only scheduled in Park rooms.

   However, if a room in Taylor satisfies the requirements for a Math class, the class can potentially be scheduled in that room to decrease conflict. It will also increase the number of rooms to eligible to hold a math class. Similarly, rooms from different buildings can be considered as possible valid rooms for different departments. This can potentially increase the capacity of some classes and decrease student conflict, consequently allowing more students to enroll in a course.

3. In the randomized data, the optimality has a positive co-relation with time slots on both constrained and unconstrained algorithms (as shown in figure 9 and 15). This may be because increasing time slots allows more flexibility in when the classes can be scheduled. This gives incentive to consider additional time slots when scheduling classes. For example, more classes can be scheduled in the 7:10 to 10 PM time slot which can potentially decrease conflict.

12 Reference Figures

12.1 Random Generated Data without additional constraints

Figures 1-5

12.2 Random Generated Data with additional constraints

Figures 6-11

12.3 BMC Data without additional constraints

Figures 12-17

12.4 BMC Data with additional constraints

Figures 18-23

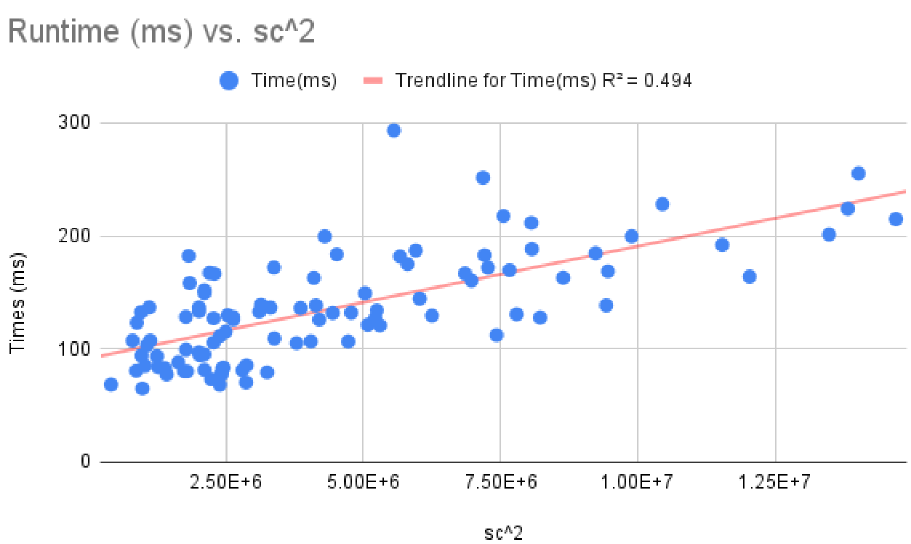

Figure 5: Runtime

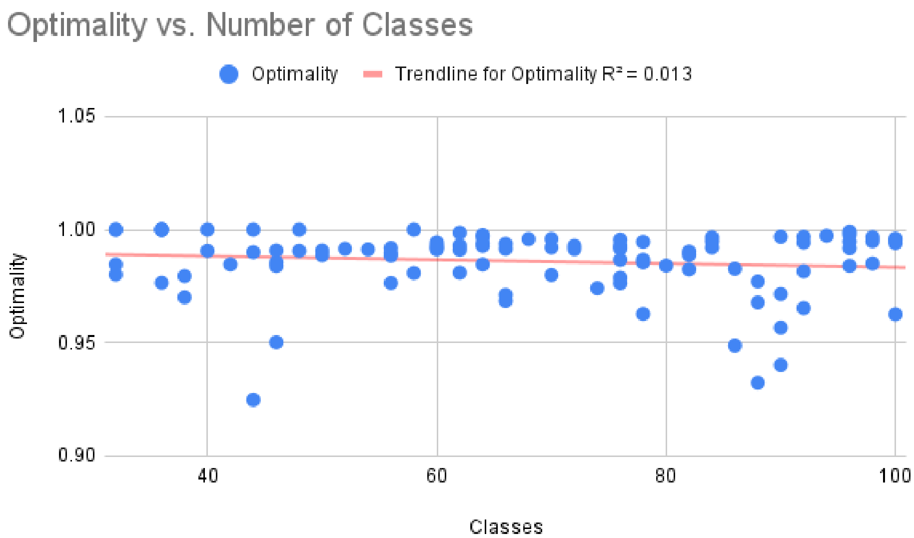

Figure 6: Classes

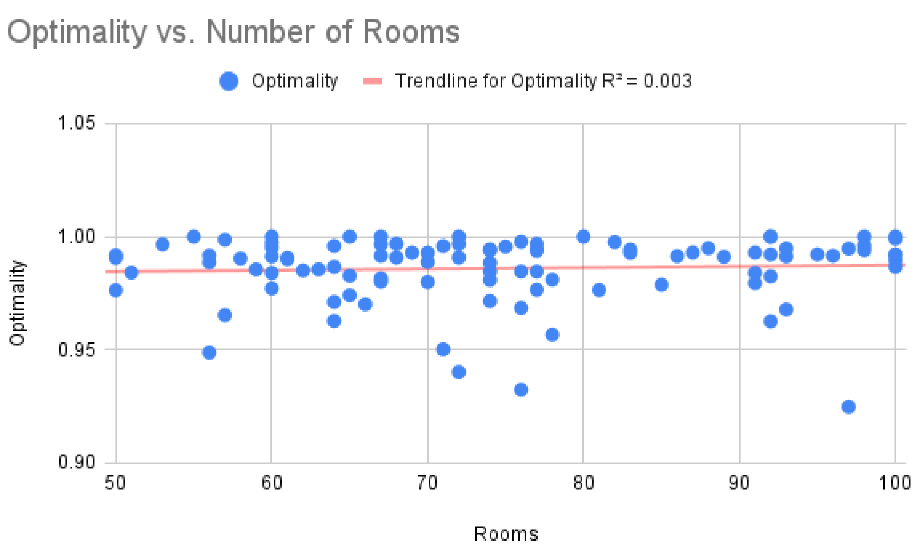

Figure 7: Rooms

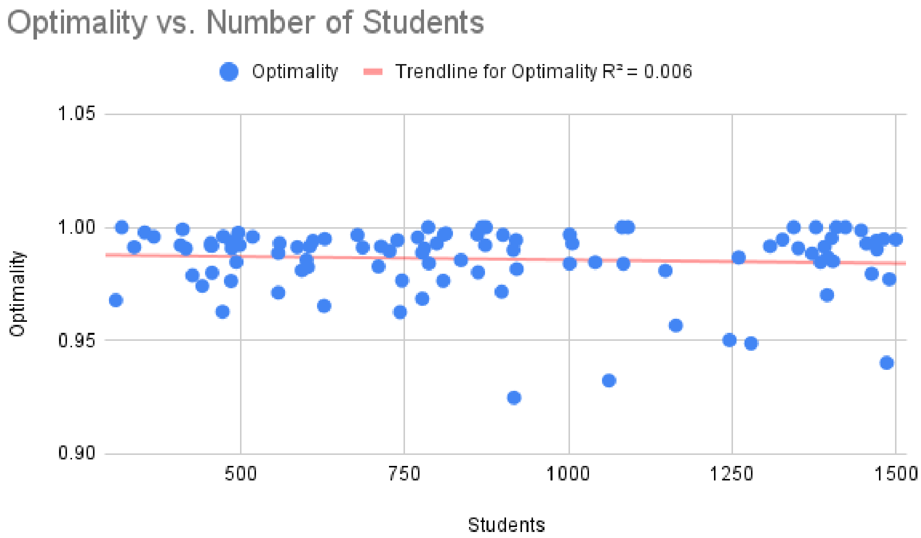

Figure 8: Students

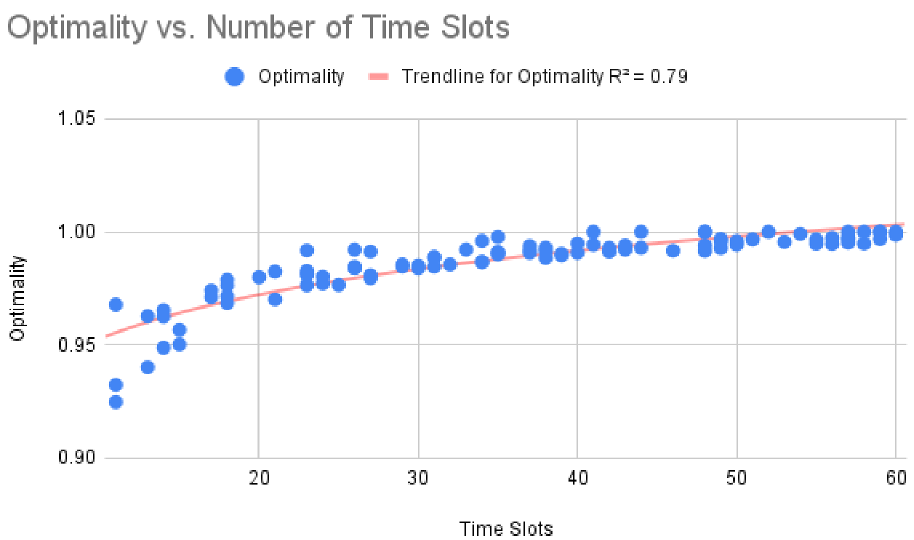

Figure 9: Timeslots

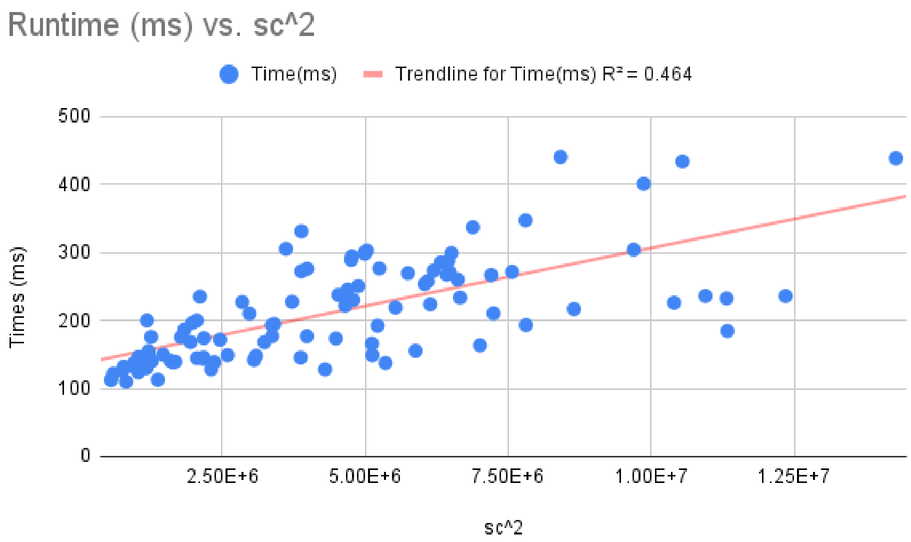

Figure 10: Runtime

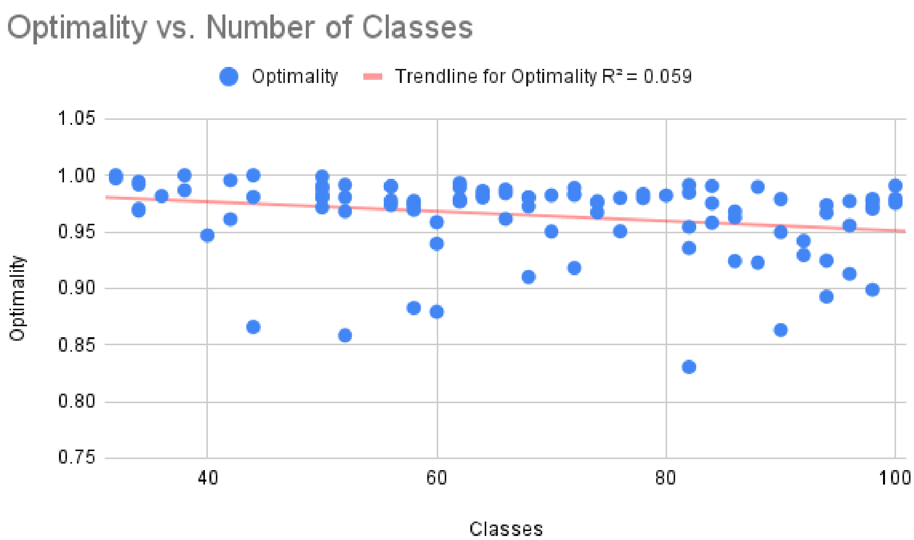

Figure 11: Classes

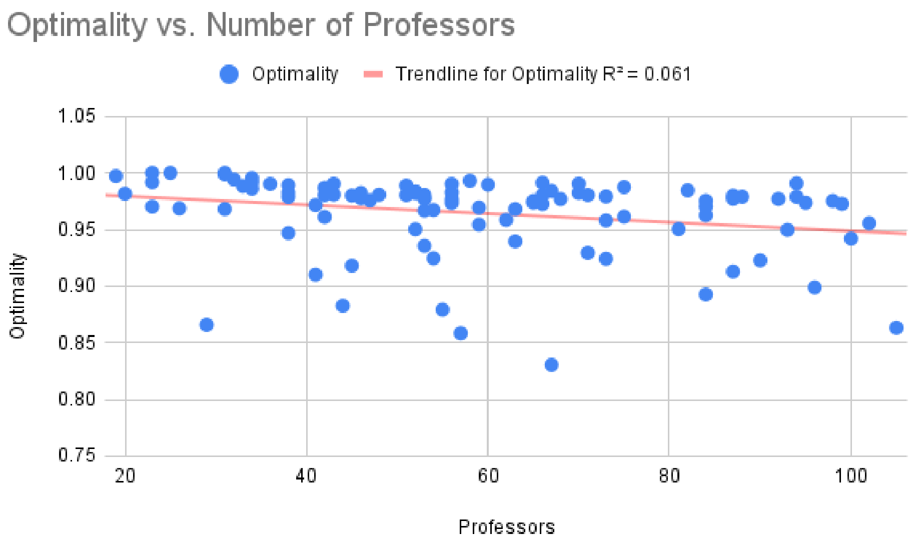

Figure 12: Professors

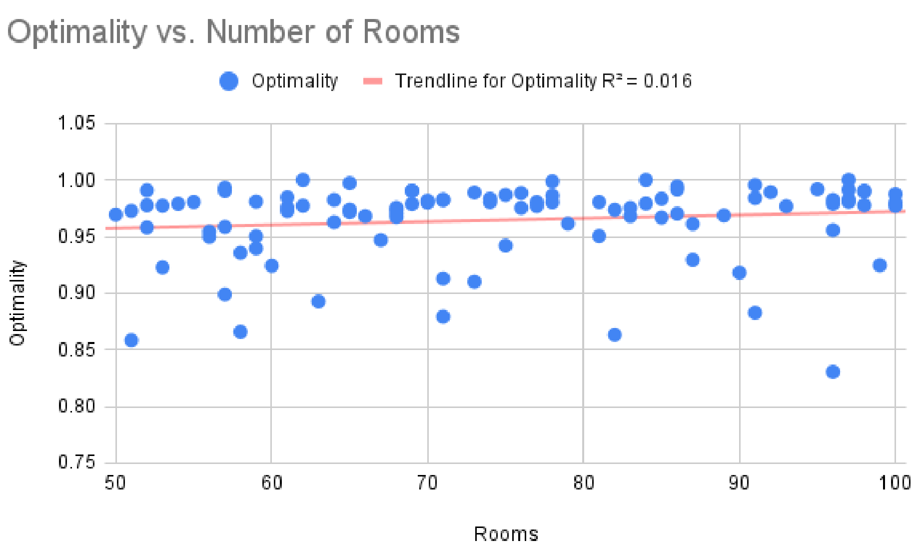

Figure 13: Rooms

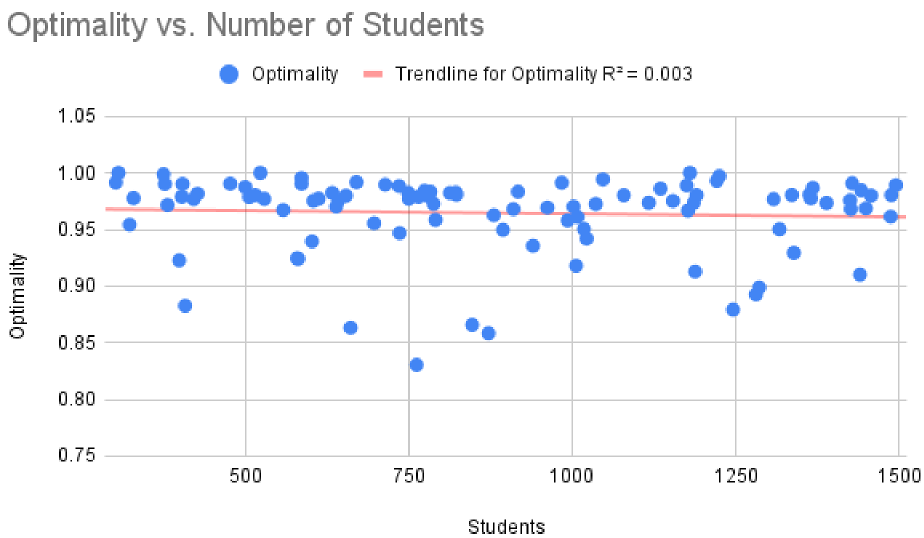

Figure 14: Students

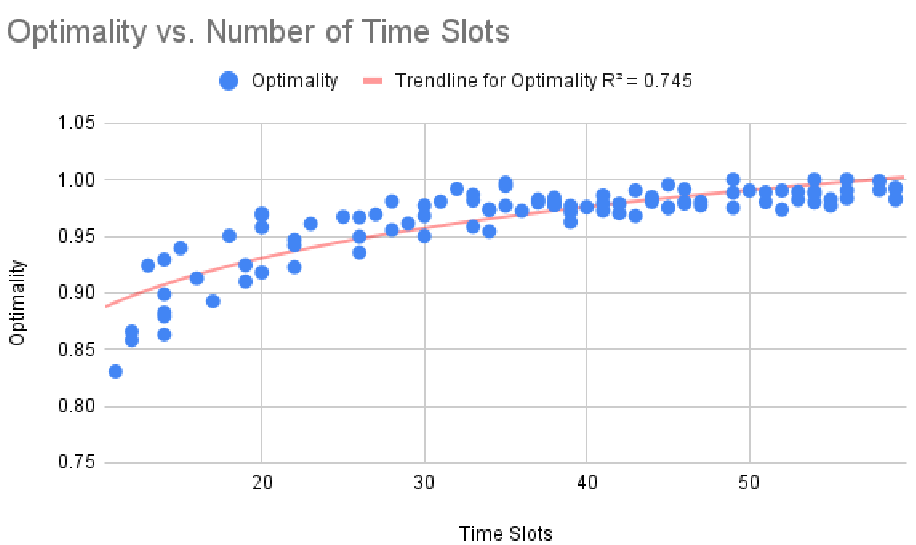

Figure 15: Timeslots

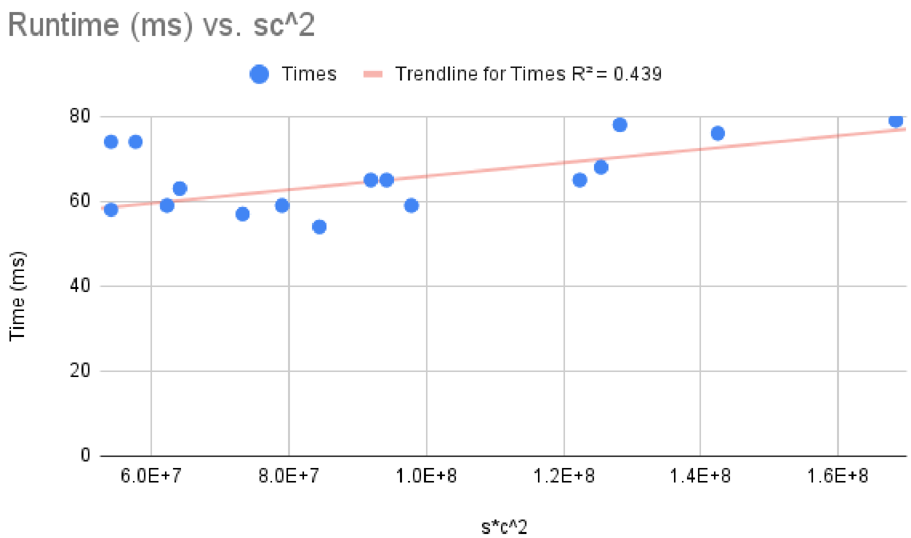

Figure 16: Runtime

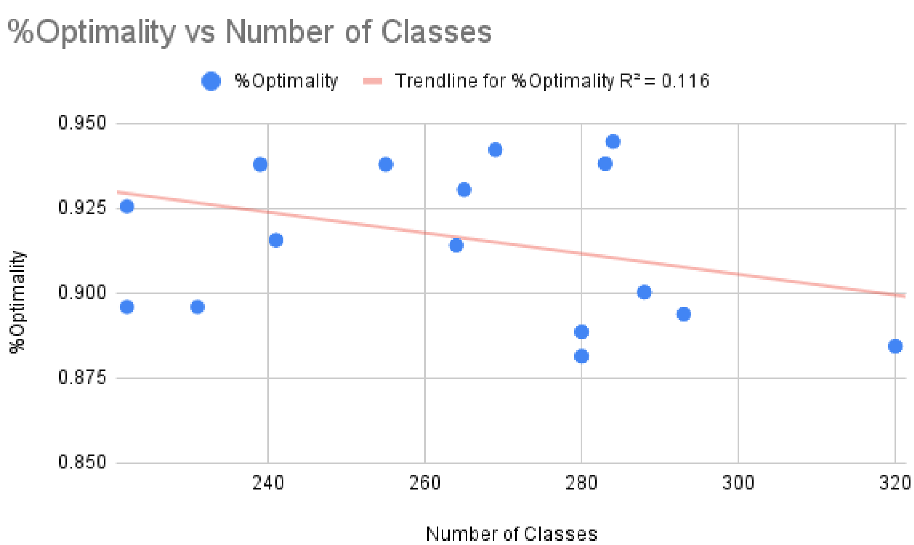

Figure 17: Classes

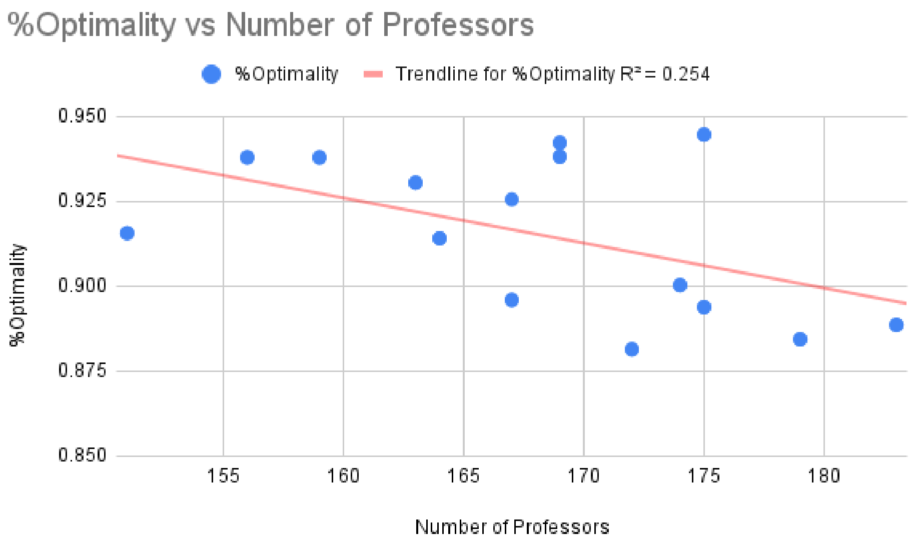

Figure 18: Professors

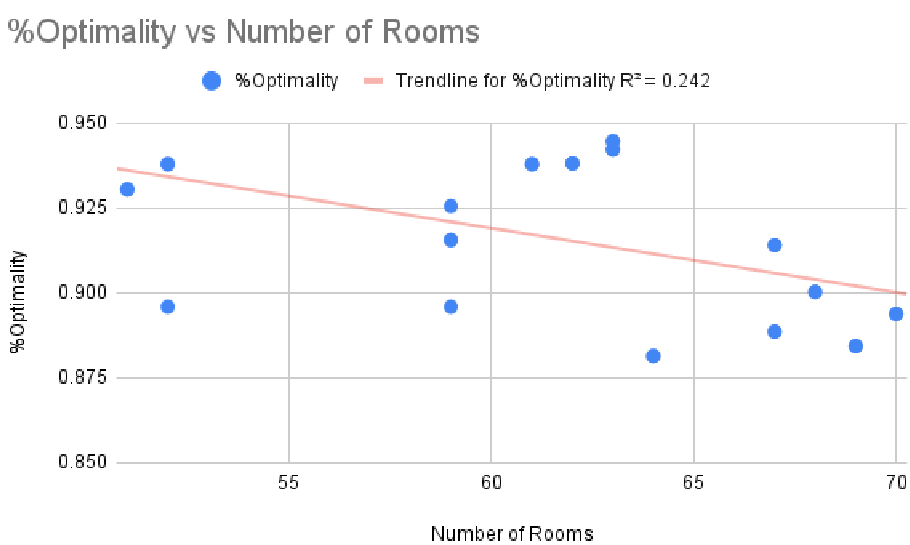

Figure 19: Rooms

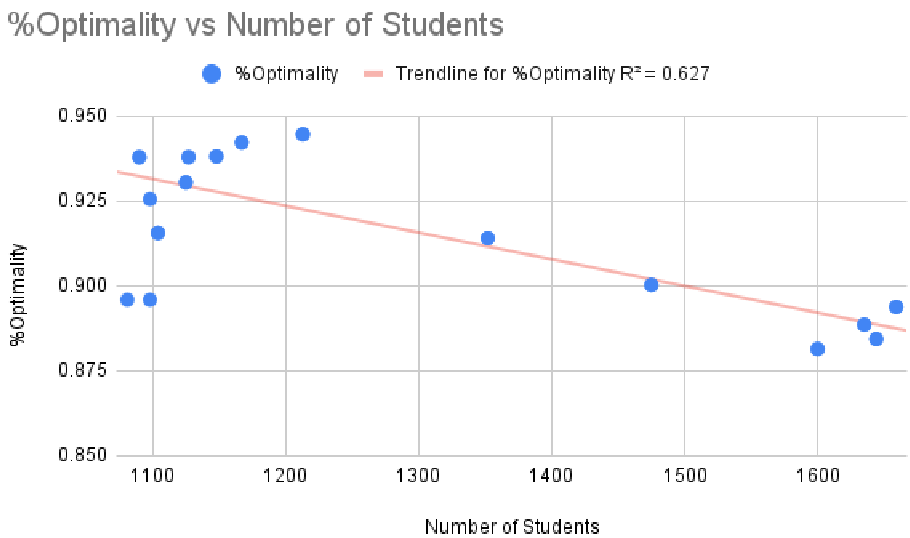

Figure 20: Students

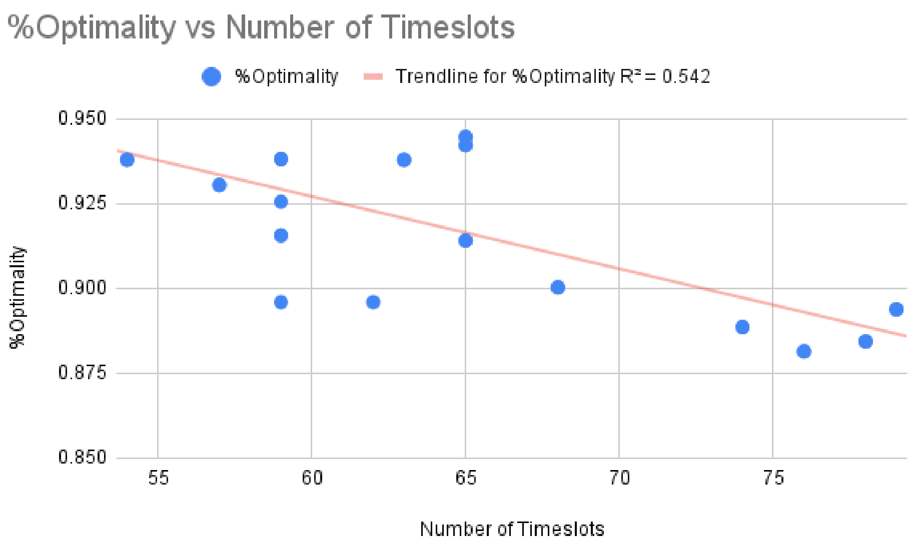

Figure 21: Timeslots

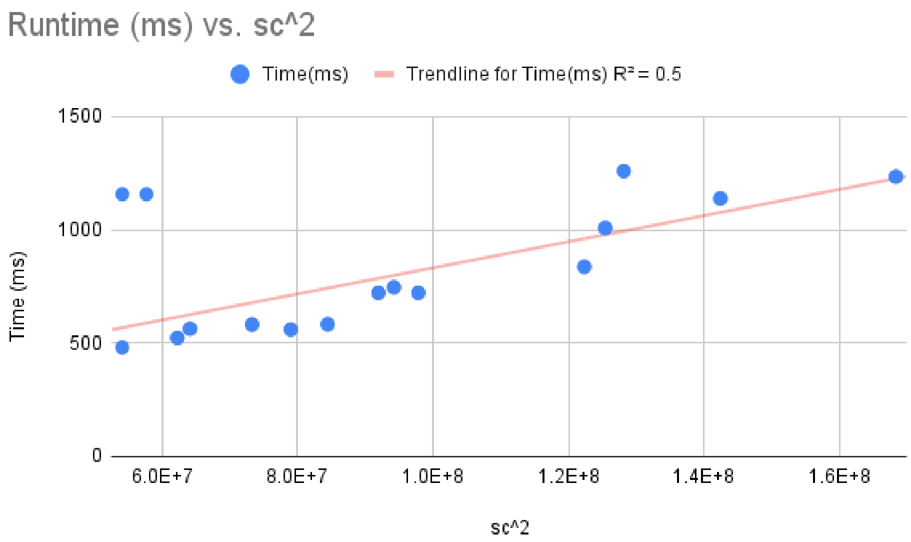

Figure 22: Runtime

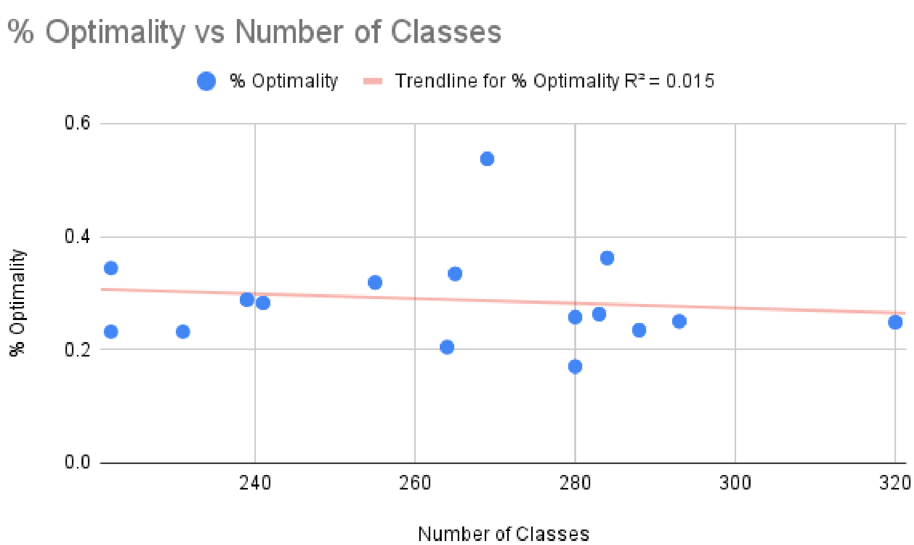

Figure 23: Classes

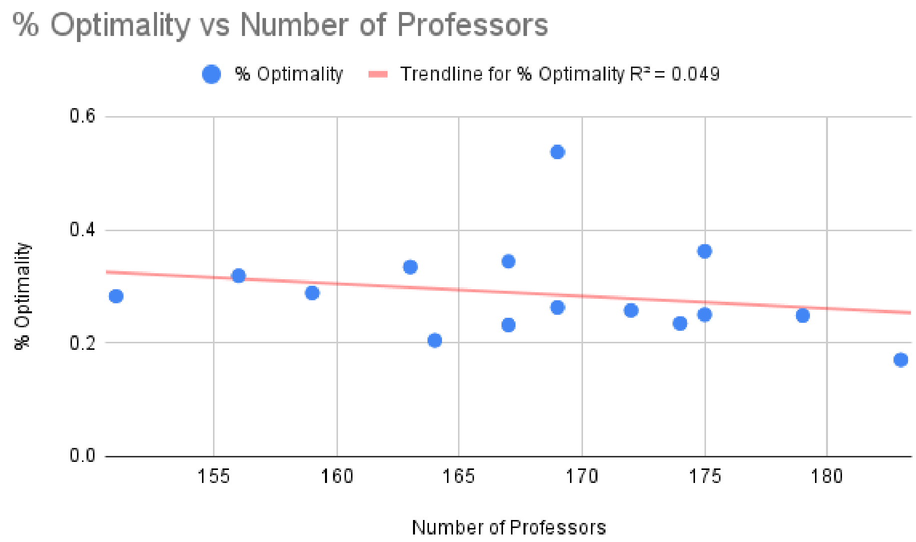

Figure 24: Professors

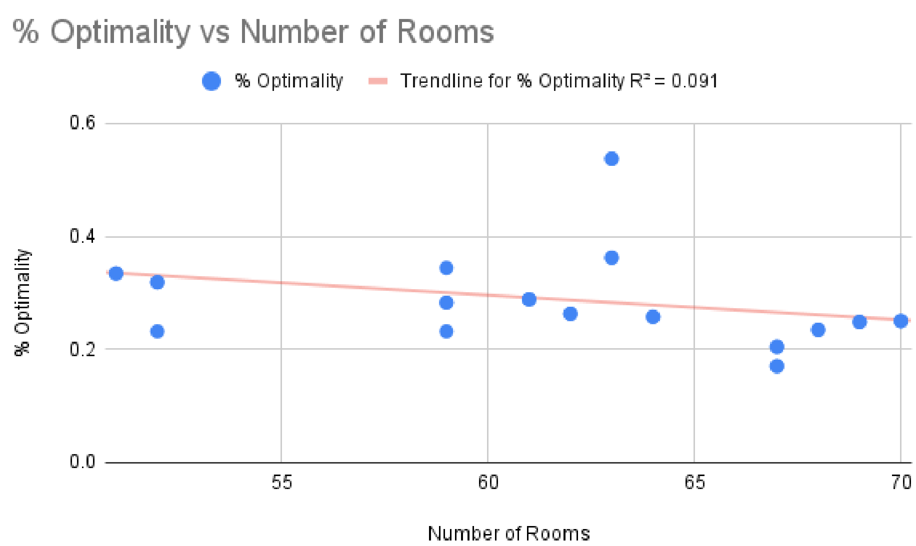

Figure 25: Rooms

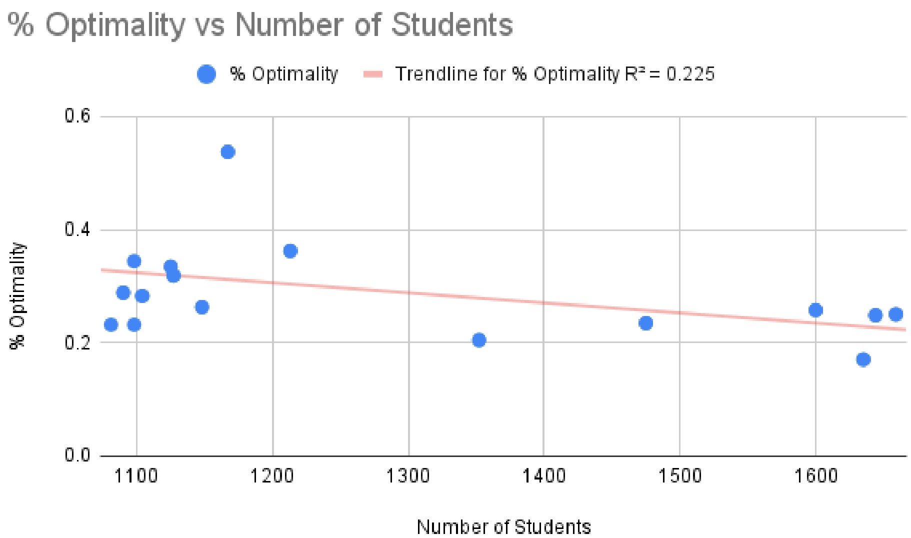

Figure 26: Students

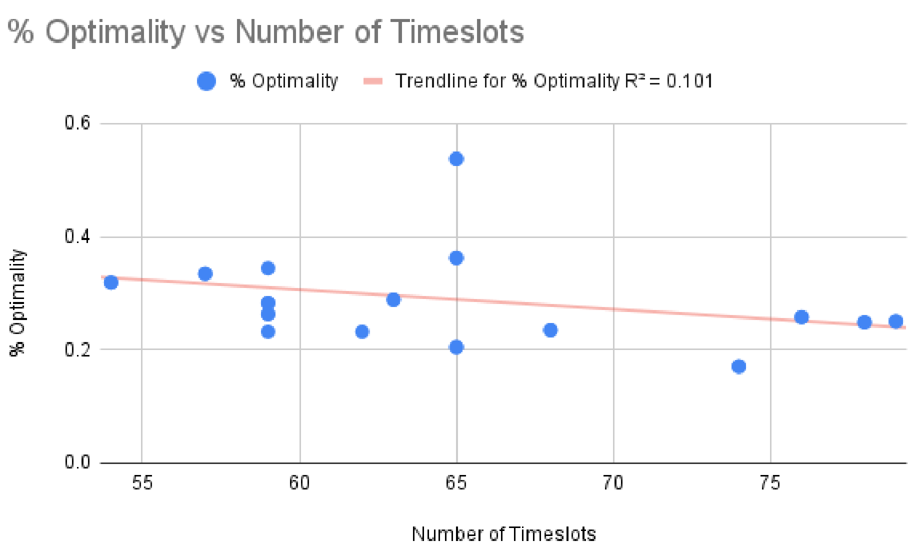

Figure 27: Timeslots