Optimizing Cornell's Final Exam Schedule

Ja Young Byun, Jolene Mei, Bennett Miller, Willem van Osselaer, Joe Ye, Eva Zhang, Jody Zhu
Advisor: Professor David Shmoys
School of Operations Research and Information Engineering

INTRODUCTION

- While **finals** are needed to test students' mastery of course contents, they are a large contributor to **stress and mental health issues** among the student population. A better final exam schedule can effectively **lower students' burdens** and grant them **more time flexibility**.
- To optimize the final exam schedule for Cornell students, we adopted methods and strategies from **combinatorial optimization**, which examines problems with a discrete set of feasible solutions. Our work was reflected for the first time in the **Spring 2022 final exam schedule**.

WHAT IS A SCHEDULE?

Monday	9:00	ORIE 6334, MATH 1920
	2:00	ENGRI 1101
	7:00	DSOC 1102, ORIE 3310
Tuesday	9:00	CS 1110, CS 2110
	2:00	HADM 4300, EAS 1540
	7:00	CRP 3348

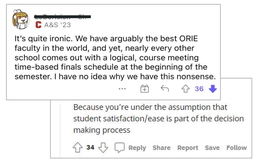


Fig.1. Left: a schedule is an assignment of exams to time slots; right: Reddit banter

- Each **combination of exams and time slots** forms a unique schedule. There are **7** days, **19** time slots, and **553** exams this semester, resulting in 19^{553} possible schedules – exam scheduling is difficult.

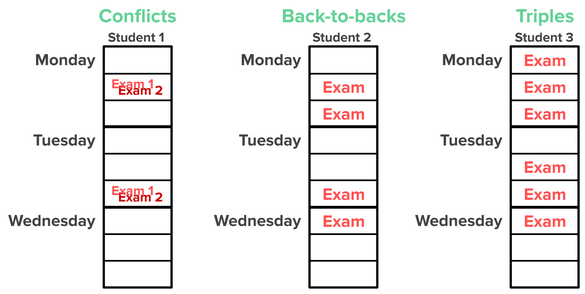


Fig.2. Examples of metrics used to determine schedule quality

- Our goal is to assign exams to time slots in a way that **minimizes** a number of metrics like the ones above. **Back-to-backs** and **triples** are further divided into subcategories that each carry a different weight.

METHODS

- We transform our data into a **graph** with nodes being the exams and edges connecting exams that share students. The edge weights correspond to the number of co-enrolled students.
- **Integer programming** is a framework for representing and solving optimization problems. Integer programs have a linear objective function and constraints that are either linear or integer.

STEP 1: BLOCK ASSIGNMENT

- First, we assign each exam to one of 19 **blocks**. Exams in the same block will be held at the same time (the particular time is not yet known). We use integer programming to find a block assignment that **minimizes conflicts**.

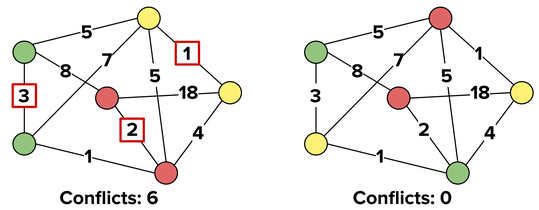


Fig.3. Two block assignments: exam nodes in the same block have the same color

Objective Function

$$\text{minimize} \sum_{i,j \in \text{exam pairs}} c_{ij} y_{ij}$$

Constraints

$$\text{subject to} \sum_{b=1}^{19} x_{ib} = 1 \quad \text{for exams } i$$

$$x_{ib} + x_{jb} \le 1 + y_{ij} \quad \text{for blocks } b \text{ and exam } i \ne \text{exam } j$$

$$x_{ib} \in \{0,1\} \quad \text{for exams } i \text{ and blocks } b$$

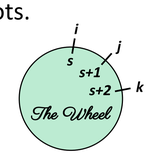
$$y_{ij} \in \{0,1\} \quad \text{for exams } i, j$$

Decision Variables

Fig.4. Basic integer program for block assignment

STEP 2: SEQUENCING

- After all exams have been assigned to a block, those blocks are then assigned to time slots. We do this by **ordering the blocks**.
- This is solved with another integer program whose objective is to **minimize back-to-backs, triples**, and a handful of other metrics. The integer program has variables tracking whether certain sequences of 3 blocks are placed in a particular sequence of slots.
- For example, $x_{ijks} = 1$ if block i is placed in slot s, block j is placed in slot $s + 1$, and block k is placed in slot $s + 2$.



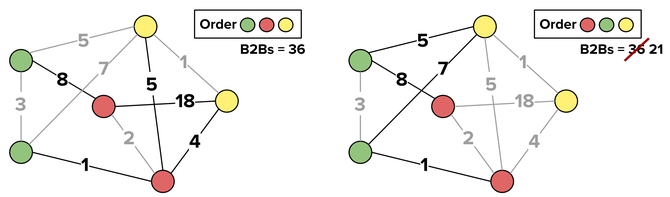


Fig.5. Finding an improved sequence given a fixed block assignment

STEP 3: POST-PROCESSING

- There can be multiple different block assignments with the same number of conflicts. Each of these solutions may lead to a very different count of back-to-backs after sequencing. Our integer program in step 1 does not necessarily return the optimal block assignment that will have the lowest number of back-to-backs (and other metrics).
- Post-processing is designed to remedy this problem in the form of a **local search**:
  - Choose an exam that is causing a lot of back-to-backs.
  - For each slot, check how many back-to-backs that exam would cause if it was assigned to that new slot.
  - Move the exam to the slot where it would cause the least number of back-to-backs, all without increasing the number of conflicts.
  - Repeat.

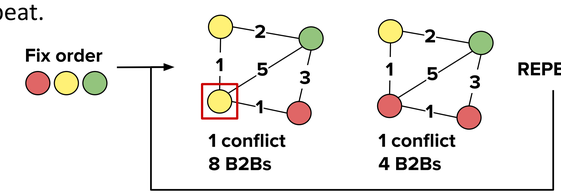


Fig.6. Diagram of local search algorithm

FINAL RESULT

- Lastly, the University Registrar requested we limit the number of large exams at the end of the final exam period, which we call **frontloading**. We frontloaded the schedule while keeping the other schedule metrics as low as possible.

	Old Model	No Frontload	Spring 2022
Conflicts	0	0	0
Back-to-backs	2609	1451	1665
Triples	84	50	76
3 in 4 slots	415	232	271
Quads	2	1	2

- Et voila! This is how the **Spring 2022 final exam schedule** was created with the help of **combinatorial optimization**.

MOVING FORWARD

- The scheduling process is still being improved for future semesters. We are experimenting with using another integer program to move multiple exams at a time during post-processing, finding a balanced min-cut among exam nodes to build a schedule in batches, and more!