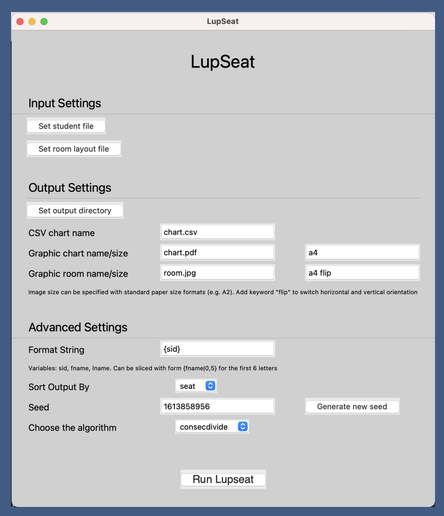
LupSeat - A Smart Seat Assignment Generator

Hiroya Gojo, Joël Porquet-Lupine
University of California, Davis. Dept of Computer Science

Introduction

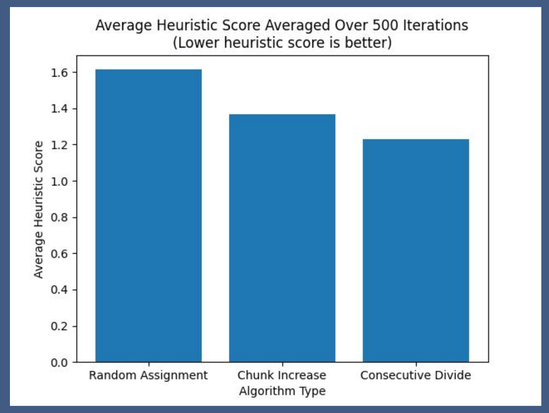
When students are allowed to sit next to whoever they want in an exam, it may increase the chance of plagiarism and cheating. On the other hand, assigning random seats to students prior to exams is time-consuming, especially for large classes, since instructors have to assign seats by hand. There are also risks of clerical errors, such as forgetting to assign a student. Existing software solutions only focus on relatively small classrooms. The goal of **LupSeat** is to automate the seat assignment process in an easy way to solve this problem. Using an algorithmic approach simplifies the process of spacing out students and minimizes the chance of errors. **LupSeat** is easy for instructors to use and instructors only need to provide student information and the representation of the room layout.

Graphical Interface



LupSeat only requires the user to specify the list of students and the layout of the room. The graphical interface, built with Tkinter, is an easy way to run **LupSeat** without having to navigate the command line. Both the GUI and command line interface provide various customizable options to tune the output exactly as needed.

Algorithms and Methodology



While developing **LupSeat**, two fundamentally different algorithms were compared, which I will call chunk increase algorithm and consecutive divide algorithm. The former is a bottom up algorithm, with the latter is a top down algorithm. The goal of the algorithms employed are essentially to minimize the size of the chunk of students so that the average number of students sitting next to each other is minimized.

The intuition with the chunk increase algorithm is to first start off with a chunk size of 1, that is, all students would have two empty seats around them. If all the students do not fit in the room with this method, the chunk size is increased by 1, and continues until all students fit in the room.

The intuition behind the consecutive divide algorithm is that the room initially starts out with the largest chunks possible, and for each empty seat, the current biggest chunk is split into smaller chunks. A backtracking algorithm was used where if the chunk needs to be split, it would first be patched together into the original large chunk, then divided again evenly.

By creating a heuristic score (average size of chunk in a room) to rank a room layout and running a script to run each algorithm many times, the **consecutive divide** algorithm was shown to consistently outperform the others.

Output Seating Chart

LupSeat is very customizable in its output. Several files are created including a graphical room layout image, a csv file with all the student/seat information, and the graphical student to seat assignment image. The image sizes are able to modified according to standard paper sizes, and the order of the students can be sorted based on their name, ID, or by their seat position. A full description of **LupSeat**'s options are available in the documentation at https://gitlab.com/luplab/lupseat.