Dorm Rooms

Room Assignment Algorithm: Approach, Policy, and Evaluation

Nisha Murali, Matthew Linsky, Cheng Wang

This program assigns students to rooms through an algorithm that calculates a **room compatibility score** based on student constraints and preferences. Critical individual constraints like **student accessibility requirements** are prioritized in scoring to ensure that students who need accessibility accommodations are assigned to rooms that have them. Additional student preferences like hall choice and building amenities contribute to personal scores in order to maximize the likelihood of satisfactory placements for as many students as possible.

The constraints supported by this program are **accessibility needs**, **room contract type** (undergraduate/graduate), and **price limitations**. These are hard parameters that must be complied with for room assignment to occur. Preferences that influence the score are residence type, hall choice, laundry and private bathroom, and roommate preference.

The chronology of the program's steps is as follows:

1. **Sorting Students by Accessibility Needs**

   The program starts by first demarcating students that need accessibility accommodations into its own separate data frame (`stu_accessdf`). Rooms with accessibility ramps are filed into its own data frame as well (`rm_accessdf`).
2. **start() function – Room Assignment Process Begins**

   The `start()` function first iterates through the accessibility needs data frames. It calls `find_room()` for each student.
3. **find_room()**

   `find_room()` receives the student object and the data frame containing the rooms it will be looking through for the student (this will be `rm_accessdf` while it is working with the accessibility needs data).

   `find_room():`

   I. Checks if the student has already found a room so as not to assign the same student to multiple rooms.
      A. If the student is already assigned a room, the function exits and returns early.
      B. If the student is unassigned, the function creates a list of their residence hall preferences (`preferred_hall_ids`).

II. Loops through the rooms of each preferred hall and uses `eval_constraints` to find a room that matches the student's constraints.
    A. If no room that corresponds to the student's constraints is found in the preferred halls, `find_room()` switches to scanning all available rooms regardless of the hall.
    B. If it passes, it checks the occupancy of that room to make sure there are available spots.

III. If there are available spots, the function assigns the student to the room.

4. **eval_pref()**

    After `find_room()` assigns a student to a room based on an initial constraints evaluation, `eval_pref()` calculates the student's actual compatibility score in respect to that room. `eval_pref()` checks if the student has a roommate preference.

5. **Roommate Preferences**

    If `eval_pref()` discovers that the student has a roommate preference, it calls `chk_mutal()`.

    chk_mutal():

    I. `chk_mutal()` checks if the roommate preference is **mutual** (both list the other as their preference). If it is, the other student is added to the room.

    II. `eval_pref()` assigns the roommate their own compatibility score for the room. An additional 2 points is added to each roommate's score for having a mutual preference.

6. **test_occupants()**

    `test_occupants()` is triggered if a student's preferred halls are filled. The function decides whether the student should swap places with a current occupant of a room. If the student's compatibility score is higher than the current occupant's, the function removes the lower-scoring occupant from the room and calls `find_room()` on the student to assign them a new room. The higher-scoring student is assigned to the room.

7. If the preferred halls are not available, then the student will begin searching for available rooms outside their hall preference using the same method as listed above.
8. After the accessibility needs students have all been assigned rooms, the program moves onto the rest of the students in the main data frames. The remaining students are **chunked** into smaller groups and then processed through the steps above.
9. Save the final room assignments to the CSV output file and export.

Main Goals & Justification

The goal of this program was to achieve high preference satisfaction while honoring all constraints. To do this, the program emphasizes accessibility and other student constraints as the determining factors in room assignment; student preferences and mutual roommate selections contribute lesser influence. This is to observe constraints and guarantee **fairness** and **practicality** in room assignments. To maintain overall student satisfaction, the program uses a deferred acceptance model to maximize the rate at which students can be provided a satisfactory room.

Some students were unable to be matched to a room because of issues related to constraints. These students were excluded from the final output to be handled separately.

The program uses numerous checks and safeguards to ensure fair room assignment and efficient program function:

- To ensure that all students are guaranteed basic suitable housing, students with accessibility needs are disaggregated and processed into accessible housing first.
- To reduce the risk of timeouts and crashes, the algorithm chunks larger student data and processes students in smaller batches through the program.
- To ensure that the same student is not assigned to more than one room, the find_room() function contains a check at the beginning to make sure that incoming students have not already been assigned scores or room placements.

The Scoring Process

A student's score for a room is calculated through the eval_pref function. The score reflects how well the room matches the student's preferences and needs. Accessibility requirements carry the heaviest weight (+9.0) to ensure that students with accessibility needs will always be assigned to a room that meets them. The scoring system is as follows:

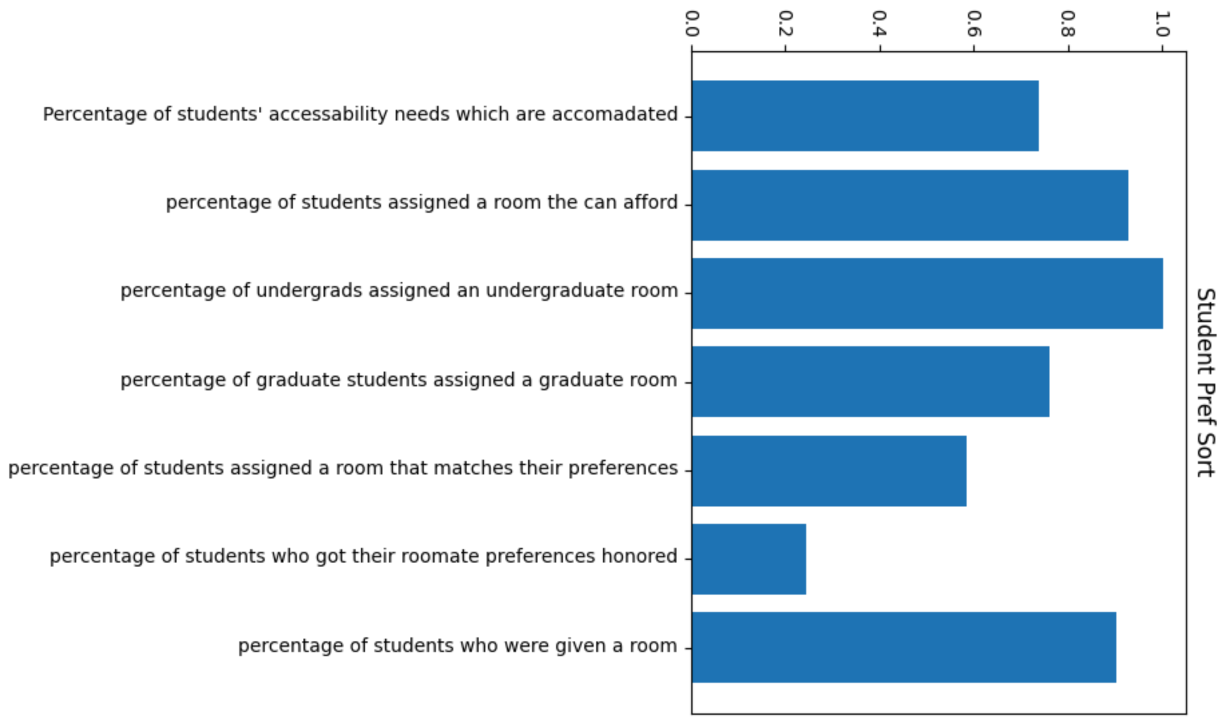
```
if room['residence_type'] ==
student['preferred_residence_type']:
#print('residence type + .5')
score += .5

if room['has_laundry'] == student['laundry_availibility']:
#print('laundry + .25')
score += .25

if room['has_private_bathroom'] ==
student['is_private_bathroom_preferred']:
#print('bathroom + .25')
score += .25

if room['hall_id'] in (student['preferred_hall_ids']):
#print('hall id + 1')
score += 1.0

if student['accessibility_need'] == 1:
#print('access + 9')
score +=9.0
```

Students with higher scores are more likely to get a room that matches their preferences because of the use of deferred acceptance through the `test_occupants()` function. A higher score signals the room is a more optimal choice for the student. `test_occupants()` uses the scoring system to resolve student collisions over room assignments. If a room is already filled up, the algorithm can replace a lower-scoring current occupant with a higher-scoring student.

Evaluation Analysis

GroupHP1_DataC's output was evaluated with Gale's evaluation program. The results are mapped in a bar chart below. 90% of the students in the data frame received a room assignment. The three key constraints adhered to by the program's matching algorithm were respected. 70% of students with accessibility needs were assigned a room that accommodated them and 90% of the overall student group were assigned a room within their price constraints. Over 90% of the undergraduate students were assigned an undergraduate room and over 70% of the graduate students were assigned a graduate room. So all constraints were satisfied with above-average results. Additionally, over 50% of students overall received a room that matched their preferences. But only just about 20% of the students had their roommate preference honored. This indicates that although the program was overall successful at meeting its target goal of complying with constraints, it possibly came at the expense of satisfying roommate selection.

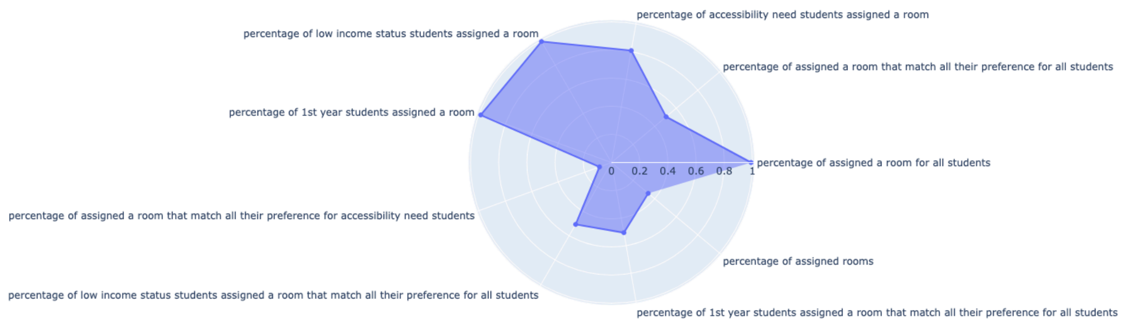
Despite this potential drawback, we feel the program is still very satisfactory. Our main goals for this program was to achieve a high level of overall satisfaction while still keeping to our constraint protocols. Our results from this evaluation demonstrate that this was achieved. More than half of the sorted students ended up with a room that matched their preferences, indicating high satisfaction with other preference metrics (such as amenities) in spite of roommate preferences being neglected. The program simply makes some sacrifices to ensure an overall successful operation.



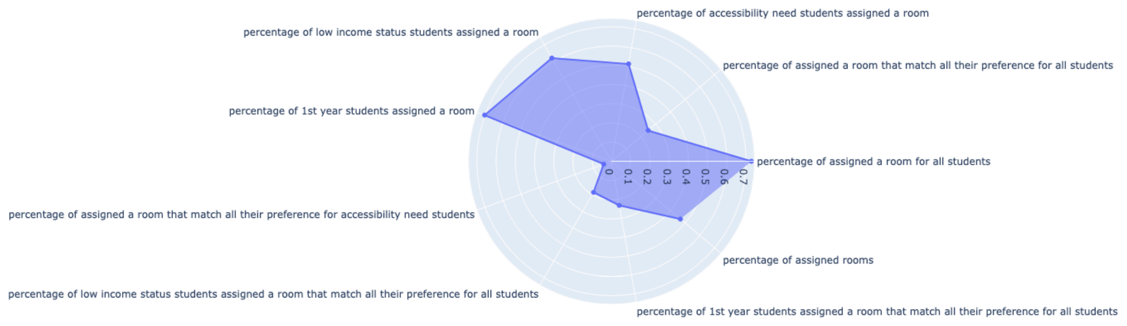
Additional Evaluation Plots

Cory's Evaluation Program:

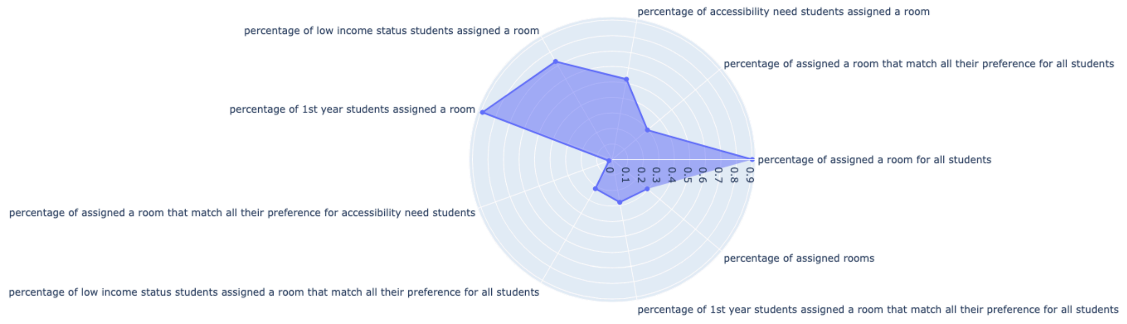
GroupHP1_DataA.csv:



GroupHP1_DataB.csv:

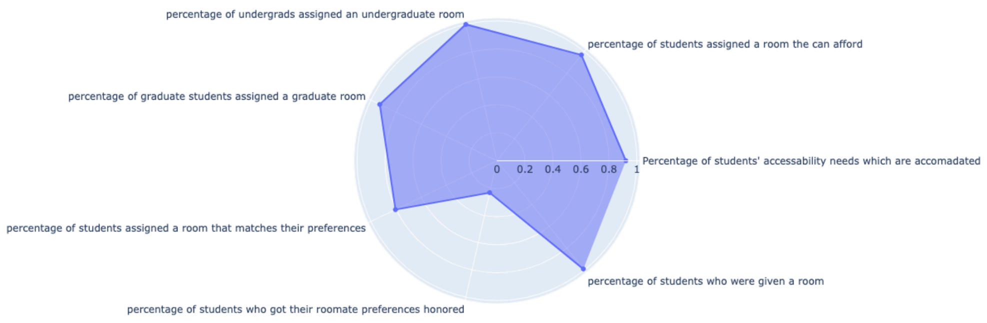


GroupHP1_DataC.csv:

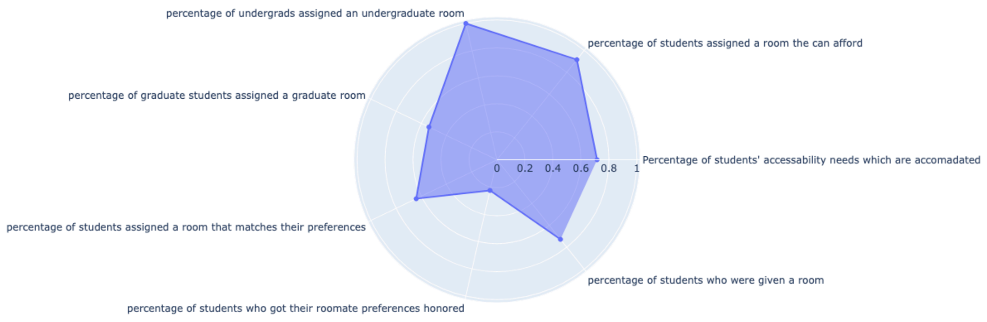


Gale’s Evaluation Program:

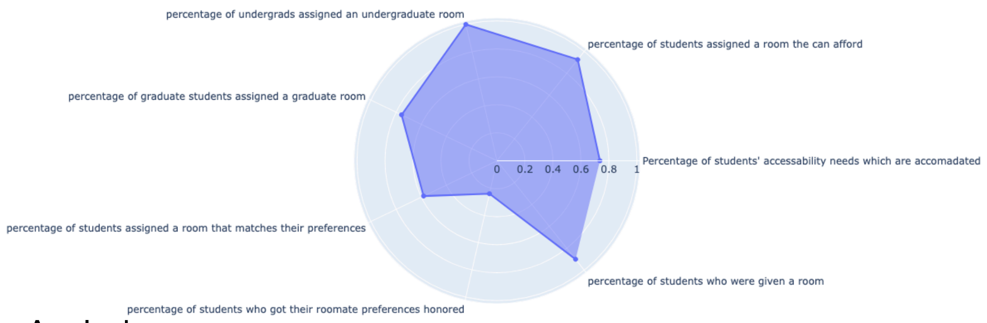
GroupHP1_DataA:



GroupHP1_DataB:



GroupHP1_DataC:



Evaluation Analysis: