Documentation Report

Anuj Shikarkhane, Jenil Shah, Yogesh Singla

November 30, 2023

1 Introduction

The optimization of the University Course Assignment System presents a unique challenge within the academic department. This document outlines the current system, identifies the challenges, and proposes potential solutions for the optimization of course assignments to faculty members.

2 System Description

Within the department, faculty members are categorized into three groups based on the course loads they handle each semester. These categories are as follows:

- **Category x1:** Handles 0.5 courses per semester.
- **Category x2:** Handles 1 course per semester.
- **Category x3:** Handles 1.5 courses per semester.

3 Unique Aspects of the Problem

The system's uniqueness stems from the flexibility it allows in the number of courses a faculty member can handle, diverging from traditional assignment problem constraints.

4 Class Breakdown

1. Professor.java :

   This class represents a faculty member with attributes like name, category, workload, and course preferences. It includes methods to check if a course can be allocated to the professor (isCourseAllocable), to allocate or deallocate a course (allocateCourse, deAllocateCourse), and to get the index of a course in the professor's preference list (getPreferenceIndex). The toString method is overridden to provide a string representation of the professor's details.

2. Course.java :

   This class encapsulates details about a course, including its name, category (like CDC or ELE), weight, and availability for allocation. It contains methods to check if the course is available (isCourseAvailable), to check if a course has been allocated to a professor of a certain category (containsCategory), and to deallocate a course from all professors (deAllocateThisCourse).

3. Category.java :

   This is a simple enumeration (enum) that defines the possible categories of professors: X1, X2, and X3.

4. Main.java :

   The Main class serves as the entry point for the application. It includes the main method where the core logic of the system is executed. The workflow involves reading faculty and course data from CSV files, categorizing faculty and courses, allocating courses based on faculty workload and preference, and finally, printing out the allocation results.

1. Initialization: The class begins by defining lists to store instances of Professor and Course objects. These lists will later be populated with data read from CSV files.

2. Reading Faculty Data: A FileReader is used to read data from a CSV file named professor.csv. A CSVParser is utilized to parse the CSV data. It is configured to treat the first record as the header. The data for each professor, including their name, category, workload, and course preferences, is read from the CSV records. For each record, a new Professor object is created with the parsed data and added to the professors list. In case of an IOException, an error stack trace is printed.

3. Categorizing Faculty: Separate lists are created for professors in categories X1, X2, and X3. The list of all professors is iterated over, and each professor is added to the corresponding category list based on their category attribute.

4. Reading Course Data: Similar to faculty data reading, course data is read from a file named courses.csv. Each course's name and type (category) are read from the CSV file. A new Course object is created for each record and added to the courses list.

5. Categorizing Courses: Courses are categorized into lists based on their type, such as courseCDC or courseELE.

6. Course Allocation Process: Courses are allocated to professors starting with category X1. The AllocateToProfCat method is called, passing the list of X1 professors and each course. If a course is still available (i.e., not fully allocated) after trying with X1 professors, it is then offered to X2 professors, followed by X3 professors. If a course remains unallocated after all attempts, it is deallocated from all professors, and the course is removed from all preference lists. This allocation process is repeated for both CDC and ELE course categories.

7. Handling Partial Allocations: After the initial allocation attempt, a new allocation round is initiated for professors in categories X2 and X3 who have a partial workload allocated (0.5 for X2, at least 0.5 for X3). This ensures that all courses have the best chance of being fully allocated.

8. Final Allocation Check: A flag is used to check if all courses have been allocated. If any courses remain unallocated, a message is printed to indicate this.

9. Printing Allocation Results: The program prints the status of course allocation (whether all courses have been allocated or not).It also prints the lists of allocated CDC and elective courses. Finally, it prints the details of all professors, including their allocated workload and courses.

10. Utility Methods: removeFromAllpreferences is used to remove a course from all professors' preference lists if it has been fully allocated.
    sortPreferencesArray sorts and returns the lowest index of a course in all preference lists, which is used to find the most preferred course that can be allocated.
    AllocateToProfCat is a method to allocate a specific course to the list of professors based on their workload capacity and preference.
    parseCoursePreferences converts a string of comma-separated course preferences into an ArrayList of course

names. This detailed process ensures that courses are allocated as optimally as possible within the constraints of faculty workload and preferences. The main method is responsible for managing the flow of this process, ensuring that each step is carried out in sequence and that the final allocations are displayed to the user.

5 Functions and Workflow of Main.java

1. Reading Faculty Data: The system reads faculty data from a CSV file and creates a Professor object for each record, storing them in a list.

2. Categorizing Faculty: Faculty members are categorized into lists based on their category (X1, X2, X3).

3. Reading Course Data: Course information is read from another CSV file, and Course objects are created and categorized.

4. Course Allocation: Courses are allocated to professors in category X1 first, followed by X2 and X3. If a course remains available after attempting to allocate to all categories, it is deallocated and removed from all preference lists.

5. Preference List Update: After allocation, if a course is no longer available, it is removed from all professors' preference lists.

6. Allocation Functions: AllocateToProfCat is a function that allocates courses to professors of a specific category based on availability and preferences. removeFromAllpreferences removes a course from all preference lists if it's no longer available.

7. Printing Results: The system outputs whether all courses have been allocated and lists the courses along with the allocation details of each professor.

8. Utility Functions: sortPreferencesArray and parseCoursePreferences are helper functions to sort preference indices and parse course preferences from a string, respectively.

The system is designed to be flexible and maintainable, with a clear separation of concerns among the different classes and methods. The Main class orchestrates the flow of data and the allocation process, demonstrating a practical application of object-oriented programming principles in solving a real-world problem.

6 Potential Modifications

To address this optimization problem, the following potential modifications can be considered:

1. Adjusting the maximum number of courses that can be assigned to faculty members within each category.
2. Introducing additional categories of faculty beyond the current three to allow for a more nuanced and generalized solution.

7 Test Cases and Results

Test 1: All courses allotted with suboptimal solution.

All CDCs provided in the courses file have been allotted to different professors. However, some professors have been allotted a workload less than their capacity. Example: a professor with capacity has been allotted a workload

of 0.5, which is half a course.

Test 2: Not all CDC's are allocated.

In this test case, all the CDC's provided are not being completely allotted which is not a fault in the logic. However, it is happening due to the constraints provided in the preference list.

Test 3: All courses allocated with optimal solution.

The courses in this test case are completely allocated to each professor in an optimal way, i.e each professor has been allocated a workload of either equal to his workload capacity or zero.

Images have been provided in the img folder.

8 Conclusion

The optimization of the University Course Assignment System is a multifaceted problem that requires balancing faculty preferences with institutional constraints. By considering the potential modifications outlined, a more efficient and satisfactory assignment system can be developed.