Discrete Structures in Computer Science (CS F222) Course Project

Nesarkar, A., Godhwani, V., Kunkolienker, R.

Fall 2023

1 Introduction

This project is prepared in partial fulfilment of the requirements of the undergraduate course Discrete Structures in Computer Science offered by the CS-IS Department, BITS Goa in the fall term of 2023.

2 Problem Statement Interpretation

The broad problem statement relates to the assignment of courses to instructors of the CS-IS Department, BITS Goa including accepting lists of course preferences from every instructor in the department, and generating valid assignments of the courses to instructors, while satisfying some constraints and also identifying – but not prescribing – the most optimal assignments based on the course preferences provided.

2.1 Specific Problem Statement

As interpreted, the specific problem is an extended version of the instructor-course assignment problem, with the following additional features: the set of instructors is divided into subsets, each of which has a characteristic maximum "course load" that each instructor in it may be assigned. For each course, there is a characteristic number of instructors the course must be assigned to. This requirement may also be satisfied by assigning a given instructor to a course more than once. The set of courses itself has four subsets: first-degree compulsory discipline courses (FD CDCs), higher-degree compulsory discipline courses (HD CDCs), first-degree elective courses (FD Elecs), and higher-degree elective courses (HD Elecs).

Additionally, each instructor maintains four ordered lists containing their preferences for courses in each category. The aim of the project is to list multiple solutions, all of which satisfy a set of constraints, and order them according to a metric that quantifies the consistency between the solution and the course-preference lists maintained by the instructors of the department.

2.2 Constraints

An assignment of instructors to courses is considered to be valid when the following conditions are satisfied:

- Every section of every CDC is completely assigned. This is given by:

$$\forall course \in CDCs, \sum_{instructor} load(instructor, course) = sections_{course} \tag{1}$$

  where $sections_{course}$ is the number of sections of *course*, each of which may be assigned to up to two instructors.

  and the *load* received by an instructor assigned to half a section of a course is 0.5

- Every elective course is assigned to either exactly the number of instructors required to take it, or to 0 instructors. No course is only partially assigned. This is given by:

  $\forall course \in Electives$,

$$\sum_{instructor} load(instructor, course) = \begin{cases} sections_{course} & \text{if } course \text{ is assigned} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

  where $sections_{course}$ is the number of sections of *course*

- No instructor receives more load from the courses assigned to them in a solution than is allowed for their category.

$$\forall instructor, \sum_{course} load(instructor, course) \leq maxLoad_{instructor} \tag{3}$$

  where $maxLoad_{instructor}$ is the maximum load that *instructor* may receive.

- There is no instructor who is assigned no courses.[1]

$$\forall instructor, 0.5 \leq \sum_{course} load(instructor, course) \tag{4}$$

[1] Enforced in Hungarian implementation. Warning on console in ILP implementation.

2.3 Objective Function

The **objective function** that must be maximized to obtain an optimal assignment is:

$$\sum_{instructors} \sum_{courses} x_{i,j} \times c_{i,j} \tag{5}$$

where $x_{i,j} = \begin{cases} 1 & \text{if instructor i is assigned to course j} \\ 0 & \text{otherwise} \end{cases}$

and $c_{i,j}$ is the cost of assigning course j to instructor i

The cost of assigning course j to instructor i is a function of the position of j on the preference list maintained by i; the higher a course is on a the instructor's preference list, the higher the cost.

3 Methods

The assignment problem has been studied at length, and there are well-known solutions that solve it in even cubic time. However, our problem statement is set apart from the ubiquitous instructor-course assignment problem by the fact that a single course may be assigned to multiple instructors, and that a single instructor may be assigned multiple courses. The constraints imposed on the assignments present additional challenges. Inspired by [2], we consider two methods for this project: Integer Linear Programming and the Hungarian Algorithm. Our proposed solutions extend to more generalised instructor-to-course assignment problems than the specific problem, as long as the input format is followed.

3.1 Linear Programming

Definition 3.1 (Linear Programming)**.** Linear Programming is defined as the problem of maximizing or minimizing a linear function that is subjected to linear constraints. Integer Programming is a linear programming variant where some or all of the variables are restricted to integers, which can be further restricted to binary case with variable either 0 or 1.

We chose to solve the linear programming problem formulated in 2.2 using the Google OR-Tools software suite [1]. Google OR-Tools is open source software for combinatorial optimization, which seeks to find the best solution to a problem out of a very large set of possible solutions. It includes solvers for Constraint Programming, Linear and Mixed-Integer Programming, Vehicle Routing and Graph Algorithms. Out of the many solvers available, we chose SCIP and CP-SAT solvers. Literature indicates that SCIP gives all intermediate solutions in the branch-cut-and-price process, however through OR-Tools we were not able to this functionality. This led us to rewrite the programme using the CP-SAT solver.

The CP-SAT solver leverages both CP and SAT approaches to efficiently handle a wide range of combinatorial problems. It formulates the problem as a set of

constraints and employs SAT-solving techniques to find a solution that satisfies these constraints. This makes it ideal for our problem statement as we want to find the feasible and not optimal solution.
The programme can be broken down into two parts.

1. Data processing
2. Solving

3.1.1 Data Processing

The code requires two files to be provided or it will fallback to the in-built defaults.

1. Courses.csv
2. Preferences.csv

Courses.csv should contain all courses offered by the department along with the type and the number of sections.

Definition 3.2 (Sections)**.** The sections of a course share the same coursework but may have different instructors assigned to them. The University may make sections for administrative purposes. E.g. CS F111 has 3 sections.

Preferences.csv should contain the category of the instructors, along with their preferences for first degree CDC, higher degree CDC, first degree elective and higher degree elective. Each preference should be on a new row and the cell should be empty if no preference.

- Data Verification
  - Ensuring Preferences Exist
    We verify that all instructors have at least one preference. If an instructor has no preferences, a warning is issued, but the code continues execution without crashing.
  - Ensuring Courses Exist
    We ensure that all courses mentioned in the preferences actually exist by comparing with the `Courses.csv` file.
  - Ensuring Sufficient Instructors
    We ensure that there are enough instructors to be assigned to all courses based on their categories. If the number of courses multiplied by the number of sections exceeds the sum of categories of all instructors, we take the following steps:
    1. Randomly remove HD electives.
    2. If the condition is still not satisfied, remove FD electives.
    3. If it is still not possible, conclude that no assignment is possible.

- Cost Matrix Creation

  We create a cost matrix where the first preferences of the instructors are given the highest weight, and those which are not in the preference list are given 0 weight. We ensure that all CDCs are represented in the cost matrix

- Output Data

  The data processing step produces the following output:

  - Cost Matrix - A matrix representing the cost of assigning each instructor to each course.
  - Category Vector - A vector indicating the category to which each instructor belongs.
  - List of Instructor Names - A list of names of all instructors.
  - List of Course Names - A list of names of all courses.
  - List of Sections - A list of sections for each course in the order they appear in the cost matrix.

Course code	**Type**	**Sections**
CS F111	FD_CDC	3
CS G524	HD_CDC	1
CS F402	FD_Elec	1
CS G519	HD_Elec	1

Table 1: Course Information

Preferences.csv and Courses.csv can be named something else but should be given as command line arguments in the correct order.

3.1.2 Solving

The decision variable is defined as shown below:

Name	Category	FD CDC	HD CDC	FD Elec	HD Elec
A Baskar	2	CS F111	CS G513	CS F429	CS G568
A Baskar	2	CS F211	CS G524	CS F430	CS F612
A Baskar	2	CS F212			
A Baskar	2	CS F241			
Aditya Challa	2	CS F211	CS G513	CS F431	CS G551
Aditya Challa	2	CS F212	CS G524	CS F432	CS G553

Table 2: Format of Preferences.csv

```
# Variables
# x[i][j] is an array of 0-1 variables, which will be 1
# if instructor i is assigned to task j.
x = []
for i in range(num_professors):
    x.append([])
    for j in range(num_courses):
        x[i].append(model.NewBoolVar(f'x[{i},{j}]'))
```

The constraints are coded as shown below.

```
# Constraints
# Each instructor is assigned to at most number of courses in
    categories.
for i in range(num_professors):
    model.Add(sum(x[i][j] for j in range(num_courses)) <=
    categories[i])

# (Optional Constraint) Each instructor is assigned minimum 1
    course
 for i in range(num_professors):
     model.Add(sum(x[i][j] for j in range(num_courses)) >= 1)

# Each half course is assigned to exactly one instructor.
for j in range(num_courses):
    model.Add(sum(x[i][j] for i in range(num_professors)) == 1)

# Maximizes the preference of assignment
model.Maximize(sum(x[i][j] * costs[i][j] for i in range(
    num_professors) for j in range(num_courses)))
```

What allows CP-SAT solver to print multiple solutions is a callback to the solver after finding one solution. Key lines of code are given below:

```
class SolutionPrinter(cp_model.CpSolverSolutionCallback):
    #refer to code for class definition

solver.parameters.enumerate_all_solutions = True
solution_printer = SolutionPrinter(x, num_professors, num_courses,
    costs, names, courses)
status = solver.Solve(model, solution_printer)
```

3.1.3 Testing results

Instructions to use the Python programme are in the Markdown documentation supplied with the code. This section of the report contains a brief description of each test case, and a description of the corresponding output with comments. Each test case has been supplied with the code, and can be used in place of the default sample by providing the input filename(s) as command-line argument(s) (see the documentation for more). Testing is done with optional constraint.

1. A typical "even" semester

   Command line arguments: *(data\Even_Sem.csv data\Even_sem_courses.csv)*

Courses: *(data\Even_sem_courses.csv)* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa. CS F111 requires six instructors, others require two.

Preferences: *(data\Even_Sem.csv)* The total available load for instructors is sufficient to assign all CDCs. Each instructor has provided 4 preferences for FD CDCs and 2 each for the other types of courses.

Output: The programme runs successfully. 2 valid solutions are printed. Values of *total cost* are high, ranging from 6420-6540. .

2. A typical "odd" semester

   Command line arguments: *data\Odd_Sem.csv*

   Courses: *(Odd_sem_courses.csv (default))* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa. All courses require two instructors.

   Preferences: *data\Odd_Sem.csv* The total available load for instructors is sufficient to assign all CDCs. Each instructor has provided 4 preferences for FD CDCs, 3 for HD CDCs and 2 each for the other types of courses.

   Output: The programme runs successfully. 2 valid solutions are printed. Values of *metric* are high, ranging from 6990-7110.

3. The preference lists are of mixed lengths

   Command line arguments: *(data\shortpreflists.csv)*

   Courses: *(Odd_sem_courses.csv (default))* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa in an even semester. CS F111 requires six instructors, others require two.

   Preferences: *(data\shortpreflists.csv)* The total available load for instructors is sufficient to assign all CDCs. Each instructor has preference lists of different lengths below 5.

   Output: The programme runs successfully and 13 valid solutions are generated. Values of *metric* are moderate to high, ranging from 5610-7140.

4. Instructors provide unavailable preferences

   Command line arguments: *data\cdc_missing.csv*

   Courses: *(Odd_sem_courses.csv (default))* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa in an even semester. CS F111 requires six instructors, others require two.

Preferences: *(data\cdc_missing.csv)* The total available load for instructors is sufficient to assign all CDCs. Each instructor has provided 4 preferences for odd-semester FD CDCs, 3 for odd-semester HD CDCs and 2 each for the other types of courses but the CDCs are not present in the list of courses.

Output: The programme runs successfully and 11 valid outputs are generated in the range 5700 - 6900. CDCs, which were not present in the preference list of any instructor, are also assigned. None of the incorrect courses are assigned to the instructors.

Explanation: Every CDC must be assigned to satisfy the constraints of the problem. Every CDC that is assigned to an instructor is not present in the instructor's preference list for CDCs, which causes *metric* to be penalised.

5. The preference lists of certain instructors are left empty

   Command line arguments: *(data\no_pref.csv)*

   Courses: *(Odd_sem_courses.csv (default))* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa in an even semester. CS F111 requires six instructors, others require two.

   Preferences: *(data\no_pref.csv)* The total available load for instructors is sufficient to assign all CDCs. Each instructor has preference lists of length 4, but they are empty.

   Output: The programme generates a warning, informing the user that a particular instructor has not filled the preference, but the code continues to run runs successfully, and 16 valid solutions are generated. The values of *metric* are moderate belonging in the range of 5670-6810.

   Explanation: Every CDC must be assigned to satisfy the constraints of the problem. But every time a course is assigned and if it is not found on a preference list and *metric* decreases.

6. The sum of categories of instructors is odd

   Command line arguments: *(data\odd_category_sum.csv)*

   Courses: *(Odd_sem_courses.csv (default))* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa in an even semester. CS F111 requires six instructors, others require two.

   Preferences: *(data\odd_category_sum.csv)* The total sum of the categories of the instructors is odd and there are sufficient number of courses.

Output: The programme runs successfully and 14 valid solutions are generated. The values of *metric* are moderate to high, belonging in the range of 5610-7140.

Explanation: However, since the sum is odd, some instructors are assigned less than their maximum capacity.

7. There are too many instructors for each to be assigned a course

   Command line arguments: *(ronit_data\toomanyprofs_input.csv)*

   Courses: *(Odd_sem_courses.csv (default))* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa in an even semester. CS F111 requires six instructors, others require two.

   Preferences: *(ronit_data\toomanyprofs_input.csv)* The minimum total load all instructors may receive exceeds the maximum total load available from all courses. Each instructor has provided 4 preferences for FD CDCs and 2 each for the other types of courses.

   Output: The programme runs successfully, and 27 valid outputs are generated, ranging from 5010 - 6360.

   Explanation: For every potential solution, there is at least one instructor unassigned as there are more instructors than the number of courses.

8. There are too few instructors to assign CDCs

   Command line arguments: *ronit_data\toofewprofs_input.csv*

   Courses: *(Odd_sem_courses.csv (default))* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa in an even semester. CS F111 requires six instructors, others require two.

   Preferences: *(ronit_data\toofewprofs_input.csv)* The maximum total load all instructors may receive is less than the minimum total load required to assign all CDCs. Each instructor has provided 4 preferences for FD CDCs and 2 each for the other types of courses.

   Output: Assignment not possible, insufficient instructors.

   Explanation: This test case is taken care in the data processing step even before the cost matrix is created. The programme exits after printing the Output message.

3.2 Hungarian Algorithm

The Hungarian Algorithm is a well-known solution to the **one-one** assignment problem. In a bipartite graph where the partitions are a set of workers and jobs respectively, and each possible edge in the graph has a "cost", the Hungarian algorithm generates a one-one bipartite matching that maximises or minimises the total cost of the matching while ensuring that all vertices in the smaller partition are connected by an edge. It is common practice[2] to model the one-one instructor-course assignment problem as a worker-job assignment problem, where the "cost" associated with an edge is a function of the position of the course represented by the sink vertex, in the preference list of the instructor at the terminal vertex.

Popular versions of the Hungarian algorithm run in quartic and cubic time. To develop our solution, we have used a Java version of the Hungarian Algorithm that claims to run in cubic time and minimises the total cost of a bipartite matching, and has very kindly been made available for free use by its author, Kevin L. Stern, in his Github repository [3].

In our project, we successively use the Hungarian algorithm to extend a typical one-one assignment problem to our more complex problem statement.

3.2.1 Definitions

Consider the following definitions in addition to those in section 2.2.

Definition 3.3 (cost)**.** For an instructor *instructor* and a course *course*, the **cost** of assigning *course* to *instructor* is defined as:
$index(course, prefList(instructor, type_{course}))$ if found in prefList(...), 10000 otherwise.

Definition 3.4 (cost matrix)**.** For a given set *instructors* of instructors and *courses* of courses, if i is the cardinality of *instructors* and c is the cardinality of *courses*, the **cost matrix** for *instructors* and *courses* is an $i \times c$ matrix whose each entry represents the cost associated with assigning the course in its column to the instructor in its row. This matrix is the sole input to the Hungarian algorithm.

Definition 3.5 (load matrix)**.** If i is the cardinality of *instructors* and c is the cardinality of *courses*, the **load matrix** for *instructors* and *courses* is an $i \times c$ matrix whose each entry represents the load received – in a given solution – by the instructor in its row from the course in its column.

Definition 3.6 (free instructor)**.** An instructor *instructor* is said to be **free** at an instant when total load they receive from all courses at that instant is less than $maxLoad_{instructor}$, the maximum total load an instructor is allowed to receive.

Definition 3.7 (vacant course)**.** A course *course* is said to be **vacant** at an instant when the number of instructors to whom it has been assigned at that

instant is strictly less than $2 \times sections_{course}$, the number of sections required for *course* to the course.

Definition 3.8 (Hungarian matching)**.** If I and C respectively are the sets of instructors and courses received as input, a **Hungarian matching** is said to be created between any subset *instructors* of I and any subset *courses* of C, when 0.5 is added to every position in the *load matrix* of I and C that lies at the intersection of a row and column between which an edge was suggested after running the Hungarian algorithm with the *cost matrix* of *instructors* and *courses* as input.

Additionally, consider the following definition for $f : I \times C \to N$

$$f(instructor, course) = \begin{cases} n - index(course, prefList(instructor, type_{course})) & \text{if } course \text{ is in } prefList(...) \\ -1 & \text{otherwise} \end{cases} \tag{6}$$

- $type_{course} \in$ {FD CDCs, HD CDCs, FD Electives, HD Electives} and is defined as the category of *Course*
- FD CDCs, HD CDCs, FD Electives, HD Electives $\subseteq$ Courses
- $prefList(instructor, type_{course}) \subseteq type_{course}$ is defined as the ordered set of courses in the preference list that *instructor* has provided for $type_{course}$
- $\forall course \in Courses, \forall C \subseteq Courses$ such that C is a zero-indexed ordered set, *index(course,C)* is the index of *course* in C
- n is defined as

$$n = max(|prefList(instructor, type_{course})|), \forall instructor, \forall type_{course} \tag{7}$$

Then *metric*, a value which identifies the more optimal of any two valid solutions, is defined as :

$$\sum_{instructor} \sum_{\substack{course \\ assignedto \\ instructor}} f(instructor, course) \tag{8}$$

3.2.2 Method of extension to problem statement

To solve the given problem statement, successive one-one Hungarian assignments have been made between subsets of the sets of instructors and courses, to arrive at an assignment that satisfies all constraints and minimises the total cost involved in each assignment step. Each final solution is modelled as a *load matrix* that indicates the load each instructor receives from a course in that solution.

Algorithm 1 Method to find a solution using successive Hungarian assignments

Require: $\sum_{instructor} maxLoad \geq 0.5 \times \sum_{course \in CDCs} reqdProfs$
Output: $LoadMatrix$

$I \leftarrow set\ of\ instructors$
$\forall P \subseteq I,\ free(P) \leftarrow \{x | x \in P,\ x\ is\ free\}$
$CDC_{FD} \leftarrow set\ of\ FD\ CDCs$
$CDC_{HD} \leftarrow set\ of\ HD\ CDCs$
$CDCs \leftarrow CDC_{FD} \cup CDC_{HD}$
$Elective_{FD} \leftarrow set\ of\ FD\ Electives$
$Elective_{HD} \leftarrow set\ of\ HD\ Electives$
$Electives \leftarrow Elective_{FD} \cup Elective_{HD}$
$C \leftarrow CDCs \cup Electives$
$\forall S \subseteq C, vacant(S) \leftarrow \{x | x \in S,\ x\ is\ vacant\}$
$\forall S \subseteq C, partial(S) \leftarrow \{x | x \in S,\ 0 < \sum_{instructor} load(instructor, x) < maxLoad_x\}$
$Loadmatrix \leftarrow load\ matrix\ of\ I\ and\ C$

while $\exists course \in CDCs$: $course\ is\ vacant$ **do**
    $create\ Hungarian\ matching\ between\ free(I) \times vacant(CDCs)$
end while ▷ CRASHES if *free(I)* becomes empty
▷ No CDC is now vacant

while $free(I) \neq \phi$ **do**
    $create\ Hungarian\ matching\ between\ free(I) \times vacant(Electives)$
end while ▷ some Electives are now partially assigned

while $partial(C) \neq \phi$ **do**
    $\forall instructor, \forall course \in partial(C)$
    if $load(instructor, course) > 0$ **then**
        $load(instructor, course) \leftarrow 0$ ▷ De-assign partially assigned courses
    end if
end while ▷ every Elective is fully assigned or not assigned

3.2.3 Output representation

Each solution is in the form a *load matrix*, an example of which is shown below.

Load matrix	CS F111	CS F222	CS G525
Instructor 1	1	0	0
Instructor 2	0.5	0	0.5
Instructor 3	0.5	0.5	0.5
Instructor 4	1	0.5	0

Table 3.2.1: An example of a load matrix for a solution.

The *load matrix* for a solution is then converted to a tabular output where each instructor is represented by a column, and every subsequent n-row sequence represents a solution, with the list of courses assigned to the instructor in that solution displayed in the column representing that instructor. Further, the value of *metric* as defined in section 3.2.1 is printed under each solution. An example of a single solution in an output file is shown below.

Instructor 1	Instructor 2	Instructor 3	Instructor 4
CS F111	CS F111	CS F111	CS F111
CS F111	CS F211	CS F222	CS F111
–	–	CS F211	CS F222

Table 3.2.2: The solution in *Table 3.2.1*, as represented in an output file.

3.2.4 Limitations

Assignments made using this method have the following limitations:

- Given two instructors with the same cost for every course, the Hungarian algorithm assigns the competing course to the instructor who appears first in the cost matrix supplied as input.
  - This has been solved by repeatedly applying the method to different permutations of the set of instructors, such that every instructor occupies every position in at least one solution.
  - However, identifying *every* equally optimal solution is not feasible as this would cause the programme to run in factorial time.
- Since there is no single objective function being minimised, there is a need to develop a makeshift *metric* (see 3.2.1) to identify the more optimal of a pair of valid solutions.

3.2.5 Testing results

Instructions to use the Java programme are in the Markdown documentation supplied with the code. This section of the report contains a brief description

of each test case, and a description of the corresponding output with comments. Each test case has been supplied with the code, and can be used in place of the default sample by providing the input filename(s) as command-line argument(s) (see the documentation for more).

1. A typical "even" semester

   Command line arguments: *none*

   Courses: *(sample_courses.csv (default))* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa in an even semester. CS F111 requires six instructors, others require two.

   Preferences: *(sample.csv (default))* The total available load for instructors is sufficient to assign all CDCs. Each instructor has provided 4 preferences for FD CDCs and 2 each for the other types of courses.

   Output: The programme runs successfully. 22 valid solutions are printed. Values of *metric* are high, ranging from 150-178.

2. A typical "odd" semester

   Command line arguments: *oddsem.csv oddsem_courses.csv*

   Courses: *(oddsem_courses.csv)* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa in an odd semester. All courses require two instructors.

   Preferences: *(oddsem.csv)* The total available load for instructors is sufficient to assign all CDCs. Each instructor has provided 4 preferences for FD CDCs, 3 for HD CDCs and 2 each for the other types of courses.

   Output: The programme runs successfully. 17 valid solutions are printed. Values of *metric* are high, ranging from 122-140.

3. The preference lists are of mixed lengths

   Command line arguments: *shortpreflists.csv*

   Courses: *(sample_courses.csv (default))* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa in an even semester. CS F111 requires six instructors, others require two.

   Preferences: *(shortpreflists.csv)* The total available load for instructors is sufficient to assign all CDCs. Each instructor has preference lists of different lengths below 5.

   Output: The programme runs successfully and 27 valid solutions are generated. Values of *metric* are moderately high, ranging from 62 to 98.

Explanation: As the length of the preference lists decreases, so does the probability of a course selected at random being found on the preference lists of the instructors it is assigned to. Consequently, *metric* decreases.

4. Instructors provide unavailable preferences

   Command line arguments: *oddsem.csv*

   Courses: *(sample_courses.csv (default))* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa in an even semester. CS F111 requires six instructors, others require two.

   Preferences: *(oddsem.csv)* The total available load for instructors is sufficient to assign all CDCs. Each instructor has provided 4 preferences for odd-semester FD CDCs, 3 for odd-semester HD CDCs and 2 each for the other types of courses but **these CDCs are not present in the list of courses**.

   Output: The programme runs successfully and 25 valid solutions are generated, but values of *metric* are very low, ranging from just 2 for the most optimal solution, to -10 for the least. **No course absent from the course file is assigned to an instructor**.

   Explanation: Every CDC must be assigned to satisfy the constraints of the problem. Every CDC that is assigned to an instructor is not present in the instructor's preference list for CDCs, which causes *metric* to be penalised.

5. The preference lists are left empty

   Command line arguments: *emptypreflists.csv*

   Courses: *(sample_courses.csv (default))* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa in an even semester. CS F111 requires six instructors, others require two.

   Preferences: *(emptypreflists.csv)* The total available load for instructors is sufficient to assign all CDCs. Each instructor has preference lists of length 4, but they are empty.

   Output: The programme runs successfully and 30 valid solutions are generated, but values of *metric* are highly negative, ranging from -54 to -62.

   Explanation: Every CDC must be assigned to satisfy the constraints of the problem. But every time a course is assigned, it is not found on a preference list and *metric* decreases.

6. There are too many instructors for each to be assigned a course

   Command line arguments: *toomanyprofs.csv*

Courses: *(sample_courses.csv (default))* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa in an even semester. CS F111 requires six instructors, others require two.

Preferences: *(toomanyprofs.csv)* The minimum total load all instructors may receive exceeds the maximum total load available from all courses. Each instructor has provided 4 preferences for FD CDCs and 2 each for the other types of courses.

Output: The programme runs successfully but no solutions are generated.

Explanation: Every potential solution fails to satisfy the constraint that the minimum load that must be received by any instructor is 0.5. Without this constraint, every solution would present at least one instructor who is not assigned any course.

7. There are too few instructors to assign CDCs

   Command line arguments: *toofewprofs.csv*

   Courses: *(sample_courses.csv (default))* All the CDCs and elective courses that may potentially be offered by the CS-IS Department, BITS Goa in an even semester. CS F111 requires six instructors, others require two.

   Preferences: *(toofewprofs.csv)* The maximum total load all instructors may receive is less than the minimum total load required to assign all CDCs. Each instructor has provided 4 preferences for FD CDCs and 2 each for the other types of courses.

   Output: The programme crashes.

   Explanation: The programme recursively attempts to assign half-sections of all vacant CDCs to free instructors using the Hungarian algorithm until there are no vacant CDCs. In this test case, the set of free instructors becomes empty before the set of vacant CDCs does, and the Hungarian algorithm is called with a zero-dimension *cost matrix*, which causes Java to throw an ArrayIndexOutOfBoundsException.

References

[1] Google AI. https://developers.google.com/optimization.

[2] Victor Keesey Fuentes. "Assigning Teaching Assistants to Courses: Mathematical Models". In: (2014).

[3] Kevin Stern. https://github.com/KevinStern/software-and-algorithms/blob/master/src/main/java/blogspot/software_and_algorithms/stern_library/optimization/HungarianAlgorithm.java.