AI505/AI801, Optimization – Obligatory Assignment 2

2026-05-04

Case 2: Team Formation

Introduction

Collaborative learning is a method for engaging learners by promoting active participation and interaction among them. It is based on the idea that learning is a social process that occurs through communication and idea exchange among team members to construct knowledge together. In practice, three approaches are employed for team formation: random grouping (i.e., assigning learners in the teams by chance), self-selected grouping (i.e., the learners choose with whom they want to work), and controlled grouping (i.e., assigning learners in the teams by instructors or computing systems based on certain criteria). In this assignment you will develop a system for controlled grouping. The goal is to assign learners to teams while maximising the overall affinity between the learners sharing the same team. To measure affinity, labels for different attributes are associated to each learner to characterise them with respect to aspects such as spoken language, gender, gymnasium attended, etc. Learners with disagreements must be assigned to different teams. To reflect the fact that some aspects are more important than others, a weight is associated to each attribute.

There is research showing that in fact groups formed by heterogeneous attributes (e.g., different skills) can be more effective than groups formed by affine attributes (e.g., similar skills) in terms of learning outcomes. However, the optimal team composition is still an open question and it may depend on several factors such as the task to be performed, the context of the learning activity, and the characteristics of the learners. Moreover, switching from one team composition strategy to the other might not be too difficult once a system for controlled grouping is in place. Therefore, in this assignment we will focus on affinity-based team formation, which is a common approach for controlled grouping.

Detailed description

We want to team up a set S of students indexed by s. Each student is characterized by a set of attributes, such as spoken language, gender, postal code, etc. We denote the set of attributes by $A = \{1..m\}$ indexed by a. We assume that all these attributes are categorical. A categorical attribute $a \in A$ takes values from a finite set of categories (or labels) $L_a = \{1..v_a\} \subset \mathbb{N}$ indexed by b. For example, the gender of a person can be mapped into the integer numbers 1 and 2. Thus, a student $s \in S$ is characterized by a vector $\boldsymbol{c}_s \in \mathbb{N}^m$, where the a-th component of $\boldsymbol{c}_s$, denoted by c_{sa}, is the label of student s for attribute a. Further, let $\omega : A \to \mathbb{Z}_0^{+m}$ be weights determinining the importance of the attributes.

We aim at combining the students in S into a set of teams $\mathcal{T} \subset 2^S$. We can denote such a team formation as a mapping $\sigma : S \to \mathcal{T}$. Thus, $\sigma(s) = T$, if student $s \in S$ is assigned to team $T \in \mathcal{T}$. We want the team formation to be a partition of S, that is, $T_1 \cap T_2 = \emptyset$ for any $T_1, T_2 \in \mathcal{T}$ and $\bigcup_{T \in \mathcal{T}} T = S$, and such that the size of each team T in $\mathcal{T}$ under σ is either $\ell = \lfloor |S|/|\mathcal{T}| \rfloor$ or $u = \lceil |S|/|\mathcal{T}| \rceil$, i.e., as equal as possible. Further, no pair of students with a disagreement listed in $D = \{(s_1, s_2) \in S \times S \mid s_1 \neq s_2\}$ can be assigned to the same

team. Among all team formations satisfying these requirements, Σ, we want to find those that minimize the total weighted sum of the different labels present in each team, thus maximizing the affinity of the members.

More formally, we want to find a team formation $\sigma \in \Sigma$ that minimizes the following objective function:

$$\sum_{T \in \mathcal{T}} \sum_{a \in A} \omega(a) z_{T,a} = \sum_{T \in \mathcal{T}} \sum_{a \in A} \sum_{b \in L_a} \omega(a) y_{T,a,b}$$

where $z_{T,a}$ is the number of different labels present in team T for attribute a, and $y_{T,a,b}$ is a binary number indicating whether there is at least one student in team T with label b (i.e., $y_{T,a,b} = 1$ iff $\exists_{s \in S} \sigma(s) = T, c_{sa} = b$).

Instance data file

The input file contains several lines with several numbers separated by white spaces. The file may contain lines with comments. Such lines start with a cardinal (`#`) and must be ignored during parsing.

- The first line contains six values: $|S|$, $|\mathcal{T}|$, m, $|D|$, ℓ and u;
- The second line contains m values representing the weights of the attributes;
- Each of the following $|S|$ lines contains the attribute vector $\boldsymbol{c}_s$ of labels associated to each of student $s \in S$;
- Each of the following $|D|$ lines contains two values, representing the indices of two students which have a disagreement.

Note: although in the description above labels and indices start at 1 in the instances and scripts provided labels and indices start at 0 as most convenient in computer science.

Solution file

The solution file must contain a single line with $|S|$ values, separated by white spaces, representing the team assignment for each student. Teams are indexed from 0 to $|\mathcal{T}| - 1$. For example, if the first value is 1, then student 0 is assigned to team 1. If the second value is 0, then student 1 is assigned to team 0, and so on.

Example

Consider the following example with $|S| = 13$ students and $|\mathcal{T}| = 3$ tables, where $\ell = 4$ and $u = 5$. There are $m = 3$ attributes.

Instance 1 (No disagreements)

```
# 13 tmembers, 3 teams, 3 attributes, 5 disagreements, 4 to 5 tmembers per team
13 3 3 5 4 5
# Attribute weight
30 20 10
# Labels associated to each tmember
1 5 2
0 7 1
1 1 1
1 1 2
0 5 3
1 7 0
1 5 3
0 5 1
```

```
0 1 0
1 0 3
0 4 1
1 2 3
1 4 1
# Disagreements
6 0
10 4
10 11
12 1
12 9
```

Solution 1

Example solution (0-based team indices):

```
1 1 0 2 2 0 1 0 2 2 0 1 2
^           ^
```

the solution is not feasible, students 0 and 6 cannot be in the same team.

Solution 2

Example solution (0-based team indices):

```
0 1 0 2 2 0 1 0 2 1 0 1 2
```

This is feasible for the size of the teams and the disagreement constraints. The first team (Team 0) has 2 different labels for attribute 0, 4 different labels for attribute 1, and 3 different labels for attribute 2. The second team (Team 1) has 2 different labels for attribute 0, 4 different labels for attribute 1, and 2 different labels for attribute 2. The third team (Team 2) has 2 different labels for attribute 0, 3 different labels for attribute 1, and 4 different labels for attribute 2. Therefore, the total weighted sum of the different labels present in each team is

$$30 \cdot 2 + 20 \cdot 4 + 10 \cdot 3 + 30 \cdot 2 + 20 \cdot 4 + 10 \cdot 2 + 30 \cdot 2 + 20 \cdot 3 + 10 \cdot 4 = 490$$

or

$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 3 \\ 2 & 4 & 2 \\ 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 30 \\ 20 \\ 10 \end{bmatrix} = 490.$$

Solution 3

```
2 1 0 2 2 0 1 0 2 1 0 1 1
```

This also passes the team-size and disagreement checks and is slightly better than solution 2. It yields an objective function value of 470.

Resources Available

In the archive asg2_2.tgz you can find the following resources to help you with the assignment.

Instances

Three instances are provided, which you can use for testing your algorithms: a toy instance `tfp_7n_2q_4l_5u_3a_1d.txt` and three more challenging instances `tfp_131n_27q_4l_5u_10a_10d.txt` and `tfp_200n_40q_5l_5u_10a_15d.txt` and `tfp_300n_60q_5l_5u_10a_40d.txt` with 131 and 200 and 300 students, respectively.

Instance generator

You are also provided with an instance generator script `team_formation_generator.py`. You can use it to generate random instances of the problem. The script takes several parameters, such as the number of students, the number of teams, the number of attributes, the number of disagreements, etc. You can run the script with different parameters to generate instances with different characteristics. For example the three instances provided are generated with the following parameters:

```
python3 team_formation_generator.py -n 13 -u 5 -a 3 -W 10 -d 5 -s 2
python3 team_formation_generator.py -n 131 -u 5 -W 10 -d 10 -s 2
python3 team_formation_generator.py -n 200 -u 5 -W 10 -d 15 -s 2
python3 team_formation_generator.py -n 300 -u 5 -a 10 -W 10 -d 40 -s 2
```

Checker script

Finally, to assess your solutions you are provided with a checker script `checker.py`. You can use it to check the feasibility of your solutions and to compute their objective value. The script takes as input an instance file and a solution file, and returns whether the solution is feasible and its objective value. You can run the script as follows:

```
python checker.py tfp_13n_3q_4l_5u_3a_5d.txt tfp_13n_3q_4l_5u_3a_5d.sol
```

Make sure that you use the checker script to validate the solutions returned by your solver and to compute their objective value.

Your Tasks

The following tasks are independent from each other and can be carried out independently.

Task 1

Model the problem as an Integer Linear Programming (ILP) problem. If you are not able to model all problem constraints, try to model as many as you can. You do not need to implement and solve the model.

Task 2

Develop and test one or more construction heuristic using the ROAR-NET API specification.

Task 3

Develop and test on or more local search heuristic using the ROAR-NET API specification. You can use the solution obtained by your construction heuristic as the initial solution for your local search heuristic or a completely random solution.

Task 4

Develop and test a metaheuristic using the ROAR-NET API specification.

Remarks

- Submit your code and a report with the results of your experiments. The report should include a description of the algorithms you implemented, the experiments you carried out, and the results you obtained. You can also include any insights or observations you made during the development and testing of your algorithms. The code must run with the only parameters the instance file and the solution file.
- Where tests are required, you can use the provided instances or generate new ones using the instance generator script. You can also create your own instances by hand to test specific scenarios. You can use the checker script to assess the feasibility and objective value of your solutions.
- You are free to design the experiments you want to carry out to test your algorithms and compare their performance.
- As a minimimal requirement for Tasks 2, 3 and 4 make sure that you report the results of your best algorithms on the three challenging instances provided. To do this, report in a table the best and median solution quality attained in the three instances in a number of runs (e.g., 10 runs) and the running time. Since running times depend on the computational environment, report also the specifics of the machine you are using (e.g., CPU, RAM, etc.).
- In Task 4, metaheuristics are asymptotic algorithms but you can terminate them at 60 seconds.