FASTT: Team formation using fair division

Jeff Bulmer, Matthew Fritter, Yong Gao and Bowen Hui

May 4, 2020

Table of contents

Motivation

Motivation

Imagine a project-based course consisting of several students that need to be assigned to teams for group projects. How should we create these teams?

- Randomly?
- Let the students decide?
- Let the professor decide?

Motivation

Can we divide up groups in a way that guarantees best possible results for each project, while also ensuring each student is satisfied?

Motivation

We need to consider:

- Projects and their requirements
- Students and their skills
- Existing relationships between students
- ALL students need to be assigned to a group

Previous Work

Team Formation

Goal: Assign **several** students to **one** project, ensuring all project requirements are met

Most notable work: Lappas et al.[2]

- Considers student skills and project requirements, as well as underlying student relationships as social network graph.
- Algorithms aim to satisfy (or *cover*) all project requirements, while minimizing *communication cost*

Team Formation

Summary:

- *One* project and its requirements
- *Several* Students and their skills
- Existing relationships between students

Issues

- Problem: Only considers one project
- Solution (?): One instance per project $\rightarrow$ can no longer guarantee full coverage for each project

Fair Allocation

Goal: Assign **several** items to **several** agents in a *fair* way

- Each agent expresses value over each item
- Fairness based on these values
- **Envy-freeness**: Resulting assignment (*allocation*) is *envy-free* when no agent could trade her set of items with another agent to get a higher total value.

Applying Fair Allocation to our problem

Fair Allocation normally considers people as agents, but this does not work in our example. Instead, we consider each project as an agent, and each student as an item.

Think of this as each project being represented by a team lead, who then picks students for the group.

Now we can now apply Fair Allocation techniques to assign students to projects. All that's missing is a value function allowing projects to express a value over each student.

Why use Fair Allocation?

Fair Allocation has two properties that are exceedingly useful in our setting.

- Many efficient algorithms to calculate *complete allocations* $\rightarrow$ every student is assigned to some project
- Fairness notions suited to a groupwork setting

Fair Allocation of Several Teams to Tasks (FASTT)

Fair Allocation of Several Teams to Tasks

We call our problem Fair Allocation of Several Teams to Tasks (FASTT). An instance of FASTT is composed of the following:

- $N = \{1, ..., n\}$ is a set of projects, each with a set of requirements r_i
- $M = \{m_1, m_2, ..., m_p\}$ is a set of students, each with a set of skills s_{m_j}
- $G(M, E)$ is a graph representing the underlying social network. A low weight on the edge between two students m, q signifies that these students would work well together

Fair Allocation of Several Teams to Tasks

- X_i is the *team* of students assigned to project i
- c_i is the *coverage* of project i, defined as the number of requirements of i that are met by students in X_i[1]

[1]Explicitly, we define coverage as $|(r_i) \cap \bigcup_{s \in X_i} |$.

Communication Cost

We additionally define *communication cost* similarly to Lappas et al. via the *diameter* of a team X_i.

The diameter of a team X_i is the longest shortest path between any two nodes in X_i. Explicitly, $d_{X_i} = \max_{m,v \in X_i}(\delta(m, v))$, where $\delta(m, v)$ is the sum of edge lengths for the shortest path between students m and v.

Value Function

The value function is a two-parameter function $u(m, X_i)$, where m is the student to be assigned, X_i the team of students currently assigned.
Consequence:

- Value of each student for any project may change whenever a student is assigned.
- However, we can more easily consider communication cost and coverage by building these into the value function

Goal

Our goal is to find a team for each project such that the following conditions are met:

- As many requirements of each project as possible are met by the skills of the students assigned to that project.
- The communication cost is as low as possible for each team.
- All students are assigned to a project.

Our Value Function

We build coverage and communication cost into our value function.

For each project i, the value of a student m is comprised of

- The number of yet unmet requirements fulfilled by that student. That is, the number of skills from m that are not yet in X_i but are in r_i,
- the communication cost of adding that student to the team X_i. That is, the current value of d_{X_i} minus the value of $d_{X_i \cup m}$

Total value of team X_i is given by the number of requirements met by students in X_i minus d_{X_i}

Complete Balance up to Order and One Item

Our primary evaluation criteria is adapted from *envy-freeness*, and its relaxation *envy-freeness up to one item (EF1)*[1].

In our setting we call these *Complete Balance up to Order (CBO)*, and *Complete Balance up to Order and One Item (CBO1)*, respectively.

Complete Balance up to Order

Let $U_i(X_i)$ be the total value for project i of students assigned to project i. Then for each pair of projects i, j, i is *balanced up to order* if $U_i(X_i) > U_i(X_j)$. If for every pair of projects i, j, i is balanced w.r.t. j, then the allocation is said to be CBO.

As EF is notoriously difficult to achieve in many cases, this extends to CBO. We therefore relax CBO to CBO1. CBO1 is met if CBO could be achieved for every pair i, j by removing up to one item from X_i or X_j.

The Multiple Round Robin Algorithm

The Multiple Round Robin Algorithm

We model our solution algorithms on Fair Allocation techniques.[2]
Idea: Perform a series of truncated selection rounds. Each round proceeds as follows

- Iterate through projects, from lowest-valued team to highest-valued team.
- Each project either selects highest-valued available student, or adds "dummy student" if no remaining student has a positive value.

Once all students are assigned, an *AdjustWinners* algorithm checks the resulting allocation for CBO1, and attempts to balance it further by swapping students.
The algorithms is presented explicitly on the next slides

[2] In particular, we adapt Aziz et al.'s[1] Double Round Robin Algorithm (DRR).

The Multiple Round Robin Algorithm

```
Input: An instance $I = (N, M, G(M, E), S, R, U)$
Output: An allocation $\pi$
while There are students left to assign do
    Order all projects in increasing order by the value of their teams
    if There are any students with positive value for any project then
        Create dummy students such that the total number of students plus
          dummy students is a multiple of the number of projects
    end
    for $i$ in projects do
        if There are no students with positive value, but there are still dummy
          items then
            Select a dummy student and assign them to the project
        else
            Select the student with the highest value for this project and
              assign them to the project
            Reorder all projects as above, and set $i$ to the project with the
              lowest team value
        end
    end
end
AdjustWinners($\pi$)
```

Algorithm 1: Multiple Round Robin with Adjusted Winners (MRR)

The AdjustWinners Algorithm

```
Input: An allocation $\pi$
Output: An allocation $\pi'$
for Each pair of projects $(i, j)$ do
    if $i$ is not CBO1 w.r.t. $j$ then
        for Each student $m$ assigned to $j$ do
            if Moving $m$ from $j$ to $i$ increases $i$'s total value from $X_i$ then
                Moved $m$ from $j$ to $i$ in $\pi'$
            end
        end
    end
end
```

Algorithm 2: AdjustWinners function

a place of mind

THE UNIVERSITY OF BRITISH COLUMBIA

MRR Runtime and Performance

In rare cases, AdjustWinners may introduce additional CBO1 errors.
$\rightarrow$ This algorithm does not provably result in a CBO1 allocation for every problem instance.
Regardless, CBO1 can be achieved in most cases, as we show in the next section.
The runtime of MRR is $O((|M| \cdot n) + (n \cdot (n-1) \cdot |M|^2))$.

Experimental Evaluation

Dataset

Data: Benchmark dataset created from snapshot of the DBLP bibliography server taken on July 1, 2019 to create a benchmark dataset. [3]

[3] Processed in the same way as Lappas et al.[2]

Algorithms

We reported performance based on

- Average running time over 500 trials
- Average team coverage over 500 trials
- Number of CBO1 allocations found in trials

As a baseline comparison, we used DRR, and the Generalized Envy Graph algorithm (GEG)[1].
Each algorithm was run on the same test cases.

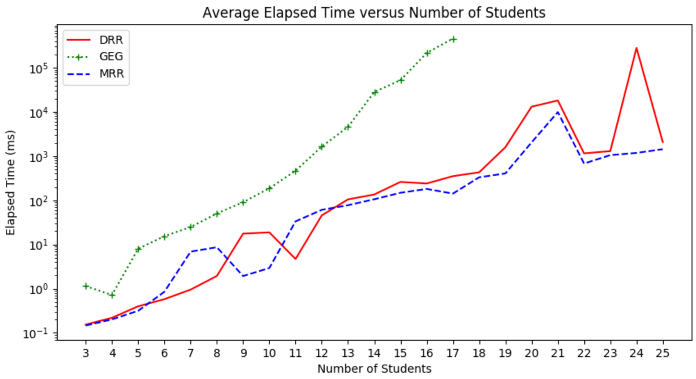

Figure: Average runtimes of each algorithm.

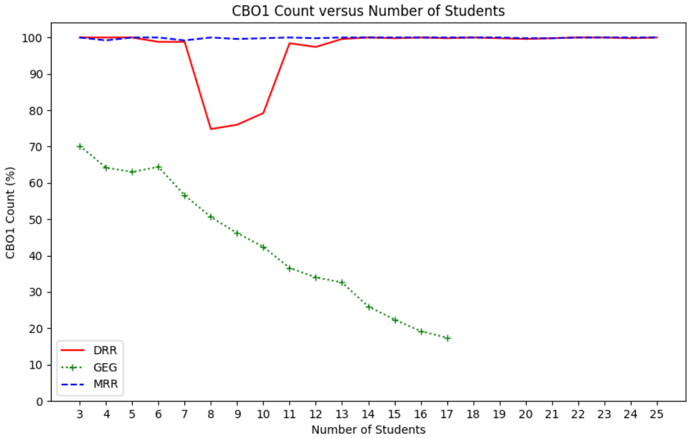

Figure: Total number of times a CBO1 allocation was found by each algorithm.

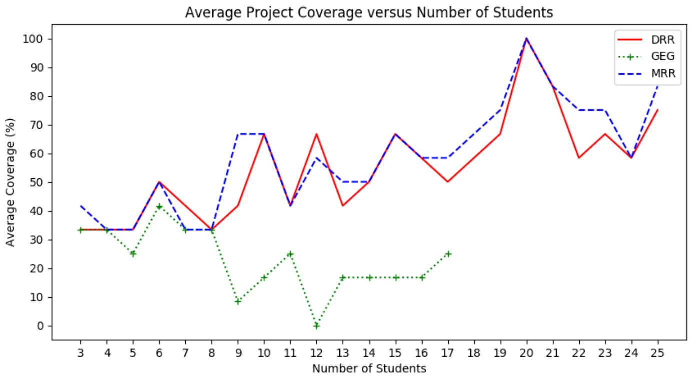

Figure: Average project coverage of each algorithm.

Implications and Future Work

Implications

Our work paves the way for future work in the fields of both Fair Allocation and Multi-Project Team Formation. In particular, we defined a setting for Fair Allocation that accounts for order, and a generalization of the Team Formation Problem that allows for multiple teams to be formed without sacrificing the benefits of single team formation.

Future Work

- Immediately, we plan to test an improved version of our algorithms on real courses
- Future work could also consider adaptations of other fairness concepts such as *proportionality* and *envy-freeness up to the least valued good (EFX)*.

References I

AZIZ, H., CARAGIANNIS, I., AND IGARASHI, A.
Fair allocation of combinations of indivisible goods and chores.
CoRR abs/1807.10684 (2018).

LAPPAS, T., LIU, K., AND TERZI, E.
Finding a team of experts in social networks.
In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2009), KDD '09, ACM, pp. 467–476.