Forming Beneficial Teams of Students in Massive Online Classes

Rakesh Agrawal
Microsoft Research

Behzad Golshan
Boston University

Evimaria Terzi
Boston University

ABSTRACT

Given a class of large number of students, each exhibiting a different ability level, how can we form teams of students so that the expected performance of team members improves due to team participation? We take a computational perspective and formally define two versions of such team-formation problem: the MaxTeam and the MaxPartition problems. The first asks for the identification of a single team of students that improves the performance of most of the participating team members. The second asks for a partitioning of students into non-overlapping teams that also maximizes the benefit of the participating students. We show that the first problem can be solved optimally in polynomial time, while the second is NP-complete. For the MaxPartition problem, we also design an efficient approximate algorithm for solving it. Our experiments with generated data coming from different distributions demonstrate that our algorithm is significantly better than any of the popular strategies for dividing students in a class into sections.

MODEL

Assume a class S of n students. Each student i is associated with *ability* $\theta_i \in \mathbb{R}$, determined using techniques such as Item Response Theory. Define the *lift* of team $T \subseteq S$, $\textsc{Lifts}(T) = \sum_{i \in T} \mathbb{I}_{\theta_i \leq \widehat{\Theta}_T}$, where $\mathbb{I}_{\text{condition}}$ is an indicator variable and $\widehat{\Theta}_T = 1/|T| \sum_{i \in T} \theta_i$. Intuitively, the $\textsc{Lifts}(T)$ is the number of students in team T that would benefit by interacting with the students of above average ability.

IDENTIFYING A SINGLE TEAM

Problem 1 (MaxTeam). *Given a set of n students $S = \{1, \ldots, n\}$, identify a team $T \subseteq S$ of at most k students such that* $\textsc{Lifts}(T)$ *is maximized.*

The `Leaders&Followers` *algorithm:* A good team consists of a set L of *leaders* of high ability who will pull up the team's overall ability and a set F of followers whose abilities will be below the team's ability yet their abilities will not be as low so as to decrease the overall ability of the team. Clearly, $\widehat{\Theta}_T$ should be larger than the largest ability score of a student in the set F thus the following *Feasibility* condition should be satisfied: $\widehat{\Theta}_T = (\sum_{i \in L} \theta_i + \sum_{i \in F} \theta_i)/k > \max_{i \in F} \theta_i$, which can be rewritten as: $\sum_{i \in F} \theta_i > k \times \max_{i \in F} \theta_i - \sum_{i \in L} \theta_i$.

Algorithm 1 `Leaders&Followers`

Input: Set of students $S = \{1, \ldots, n\}$ with sorted abilities $\theta_1 > \theta_> \ldots > \theta_n$.
Output: Team T with the maximum $\textsc{Lifts}(T)$.

```
T = ∅
for i = 1 ... k do
    L = top-i ability students
    for j = i + 1 ... n − (k − i) + 1 do
        F = students with abilities θ_j, θ_{j+1}, ... , θ_{j+k−i−1}
        if F and L satisfy feasibility condition then
return T = L ∪ F
```

If we knew the number of students in set L and the top (highest ability) student in set F (with ability score denoted as θ_F), then we can compute the right hand side of the above inequality. However, we do not know which student is going to be the top student in set F but there are only $O(n)$ possibilities. For a given set L and the top student with ability θ_F, the easiest way to satisfy this inequality is by placing the top students with ability lower than θ_F in the set F. These students are clearly a set of *consecutive* students with ability levels below θ_F. With a preprocessing step of complexity $O(n)$ for computing the cumulative sums of all the ability levels of all the students sorted in decreasing ability level, `Leaders&Followers` algorithm has complexity $O(nk)$. This complexity reduces to $O(n \log k)$ by replacing the linear search in Algorithm 1 with a binary search.

PARTITIONING A CLASS INTO TEAMS

Problem 2 (MaxPartition). *Given an integer k and a set of n students $S = \{1, \ldots, n\}$ (with $n = k\ell$) find a partition of S into teams $T_1, \ldots T_\ell$, where each team is of size k and $\sum_{i=1}^{\ell} \textsc{Lifts}(T_i)$ is maximized.*

Lemma 1. *When $k = n/2$ then the* MaxPartition *is NP-complete.*

The `IterL&F` *algorithm:* The `IterL&F` algorithm solves the MaxPartition problem by iteratively picking one team of size k using the `Leaders&Followers` algorithm in every iteration and removing this team from the set of students. Among the many candidates that may possibly exist that achieve the same value of Lifts, the one with the highest highest-ability follower is picked. Intuitively, this tie-breaking rule groups the highest ability leaders

L@S'14, March 4–5, 2014, Atlanta, Georgia, USA.
ACM 978-1-4503-2669-8/14/03.
http://dx.doi.org/10.1145/2556325.2567856

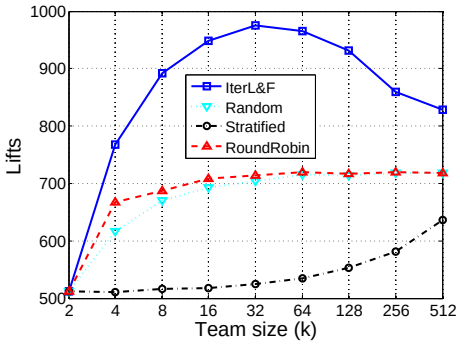

Figure 1. Performance of the `IterL&F`, `Random`, `Stratified` and `RoundRobin` algorithms for the MaxPartition problem; x-axis (log-scale): number of students per team (k); y-axis sum of the Lifts values of the teams in the partition.

with relatively high ability followers. Clearly, there are n/k iterations of `IterL&F`, each taking time $O(n \log k)$. Therefore, the overall running time is $O(n^2 \log k/k)$.

EXPERIMENTS

We experiment with a dataset of $n = 1024$ students, with ability values randomly sampled from a pareto distribution having the shape parameter equal to 3.

Baseline algorithms: In addition to the `IterL&F` algorithm, we experiment with the following three baseline algorithms: `Random`, `Stratified`, and `RoundRobin`.

The `Random` algorithm simply creates ℓ teams of size k by randomly assigning students to teams. The running time of the `Random` algorithm is $O(n)$, since it is adequate to create a random permutation of the students and then create the ℓ teams by considering consecutive members of this permutation. Note that `Random` is the frequently used algorithm for partitioning a large class into sections.

The `Stratified` algorithm sorts the students in decreasing order of their abilities. Then, the first team is created by considering the first k students with the highest abilities and putting them in a team by themselves. The second team is created with the subsequent k students and so on. The running time of this algorithms for a sorted input consisting of n students is $O(n)$. This algorithm can be thought of as an idealized version of the oft-used, ability-based homogeneous grouping of students.

The `RoundRobin` algorithm again considers the sorted list of students. In this case, the first team is created by considering k students at positions 1, $k+1$, $(2k+1)$ etc. in this sorted list. The second team is formed by students at positions 2, $k+2$, $(2k+2)$ etc. on the same sorted list and so on until ℓ teams are formed. The running time of this algorithms for a sorted input consisting of n students is $O(n)$. This algorithm mimics how teams are often formed (particularly in recreational sports) by first selecting the leaders and then letting the leaders take turn in adding members to their respective teams.

Performance of team-formation algorithms: Figure 1 shows the total gain of the teams formed by the different algorithms as a function of the team size k. The results (which are averages over 20 random datasets drawn from the respective distribution) demonstrate that for all values of k (except for $k = 2$, where all algorithms have similar performance), `IterL&F` is significantly better. In fact, there are values of k (e.g., $k = 32$) for which the `IterL&F` achieves total Lifts of more than 950, while the maximum possible value is less than 1024. This means that more than 90% of the students are assigned into teams that can potentially improve their performance since the team's ability is higher than the students' abilities for all these students.

Amongst the baseline algorithms, `Random` and `RoundRobin` are better than `Stratified`. The reason is that the dataset drawn from a pareto distribution has a small number of exceptionally high-ability students. The `Stratified` algorithm puts these students together in one team and therefore their high abilities cannot be leveraged to lift up the average abilities of other teams. This phenomenon is not observed in the teams formed by `Random` and `RoundRobin` since these algorithms distribute the high-ability individuals into different teams allowing more teams to benefit from them.

For $k = 2$, all algorithms have the same total Lifts, which is equal to $n/2 = 1024/2 = 512$. This is because in teams of size 2 inevitably there is one student that is above and one that is below average and therefore the team is beneficial for exactly half of the students, independently of how the team assignment is performed.

DISCUSSION

One could object that our approach is unfair to strong students. We offer the following counterpoints:

- Our algorithm builds teams in such a way that the highest ability leaders are grouped with the highest ability followers, the next set of highest ability leaders are put together with the next set of highest ability of followers, and so on. Thus the strong students are still in good company.
- Helping someone else understand the material can produce a more organized cognitive structure than only trying to learn the material for oneself. The person starts seeing the issues from new perspectives, leading to a better fundamental grasp of the material.
- A student could be very strong in one subject, but not so strong in another. It is only fair that she helps her team mates in her strong subject, while she gets help from them in the subject that is not her strong suit.

In the future, we would like to investigate the implication of extending the partitioning objective to incorporate the numeric increases in the performance of the group members. We would also like to enrich our problem formulation with constraints due to socio-emotional factors such as interpersonal relations. Finally, we would like to partner with some MOOCs to study the performance characteristics of our proposal in real-life settings.