Max-min fair tutor-student assignment under lexicographic preferences

MegaMath4SmallBiz

March 27, 2018

1 Scenario

Let S, T be the set of students and tutors, respectively. For each student/tutor pair (s, t) there exists a score $r_{s,t}$ that measures the affinity between student s and tutor t.

Each tutor t can mentor at most N_t students. On the contrary, there is no bound on the number of tutors that can mentor the same student. More formally, let $x_{s,t} = \{0, 1\}$ be the binary variable that equals 1 if and only if tutor t is assigned to student s. Then,

$$\sum_{s \in S} x_{s,t} \leq N_t, \qquad \forall\, t \in T. \tag{1}$$

1.1 Related works

If each tutor can only mentor exactly one student ($N_t = 1$ for all $t \in T$) then our problem is similar to the one studied in [2] and [3].

2 Desiderata

We wish to find a tutor-student assignment x^* with the following desiderata:

1. x^* must respect constraint (1), i.e., each tutor t cannot supervise more than N_t students;
2. x^* must take into account that student preferences are not linear in the affinities r with the assigned tutors. For instance, a student s arguably prefers to be assigned to a single tutor t with score 10 ($r_{s,t} = 10$) rather than to two tutors with score equal to 5 each;
3. x^* must be somehow "fair", thus it should intuitively avoid stark disparity among students. For instance, attempting to maximize the sum of affinities over all students and tutors is in general a bad idea since it does not guarantee fairness.

3 Preliminaries: max-min fairness

We first observe that attempting at maximizing the total reward over all students is in general not a good idea since it generates stark inequalities among students. In fact, the solution in this case is to let each tutor t choose the N_t students with the highest affinity. In other words, each tutor t ranks the students according to decreasing score $r_{s_1,t} \geq r_{s_2,t} \geq \dots$ and selects the first N_t of them. However, there may be students with very low reward; potentially, there may be students s with no tutors is assigned to them.

For the reason outlined above, we aim at finding a "fair" assignment between students and tutors. To this aim, let us first define the crucial concept of max-min fairness.

Definition 1 (Max-min fair allocation)**.** *[4] An allocation a^* is max-min fair when the only way to increase the reward a^*_s of a student s is by decreasing the reward of a students s' with lower reward, i.e., such that $x^*_{s'} < x^*_{s'}$.*

We remark that the max-min fair allocation can be equivalently defined as the one achieving the lexicographic sorted optimum[1].

Max-min fairness always give priority to students with smaller affinity with tutors (that would end up with a poor reward if we simply maximized the sum of rewards), and try to maximize their reward in an iterative fashion.

4 Proposed solution: Iterative Max-Min

We now describe our solution to assign fairly tutors to students, by exploiting the concept of max-min fairness.

The first idea is to apply the concept of max-min fairness to the sum of affinities between each student and all the tutors assigned to her. On the other hand, this would lead to a solution that does not well adapt to desiderata 2.

Therefore, we propose an algorithm that iteratively assigns one tutor to each student in a max-min fashion, as long as the constraints on the maximum number of students supervised by the tutors is fulfilled. By assigning only one tutor at the time, we tend to ensure high individual affinity for all students for the very first few iterations, which is in line with desiderata 3.

The main procedure can be resumed as follows.

[1] Let $\overline{R}$ be the vector R sorted in increasing order. We say that R is preferred to R' if $\overline{R}$ is lexicographically greater than $\overline{R}'$, i.e., $\exists\, n$ such that $\overline{R}_i = \overline{R}'_i$ for $i = 1, \dots, n$ and $\overline{R}_{n+1} > \overline{R}'_{n+1}$.

Algorithm 1: Iterative Max-Min fair tutor-student assignment

Data: Students S, tutors T, student-tutor score $r_{s,t}$
Result: Max-min fair student-tutor assignment τ.
$\tau(s) \subset T$ is the set of tutors assigned to student s, for all $s \in S$.

Initialization: Set $\tau(s) = \emptyset$ for all $s \in S$ and $S'' = \emptyset$, containing the set of unassigned students.
while $\exists\, t : N_t > 0$ **do**
  Set $S' := S \setminus S''$.
  while $(S' \neq \emptyset) \wedge (\exists\, t : N_t > 0)$ **do**
    Use dichotomy search Algorithm 2 to solve the following Core Problem CP:

$$\begin{cases} \widetilde{x} = \arg\max_x y \\ \text{s.t. } \sum_{t \in T_s} x_{s,t}\, r_{s,t} \geq y \qquad \forall\, s \in S' \\ \quad \sum_{t \in T_s} x_{s,t} \leq 1 \qquad \forall\, s \in S' \\ \quad \sum_s x_{s,t} \leq N_t \qquad \forall\, t \in T \\ \quad x_{s,t} \in \{0,1\}, \qquad \forall\, s \in S', t \in T. \end{cases} \tag{CP}$$

    if CP is unfeasible **then**
      Let S^u be the set of unassigned students. Set $S' := S' \setminus S^u$ and $S'' := S'' \cup S^u$.
    else
      Let $A = \{(\underline{s}, \underline{t})\}$ be the assignments produced by Algorithm 2. Assign the tutors to the students accordingly: $\tau(\underline{s}) := \tau(\underline{s}) \cup \underline{t}$, $\forall\, (\underline{s}, \underline{t}) \in A$.
      Update the set of students yet to be assigned to a tutor: $S' := S' \setminus \{\underline{s} : (\underline{s}, \underline{t}) \in A\}$.
      Update the number of available slots for each tutor $\underline{t}$: $N_t := N_t - |\{\underline{s} : (\underline{s}, \underline{t}) \in A\}|$.
      Update the set of available tutors for each student: $T_{\underline{s}} := T_{\underline{s}} \setminus \underline{t}$, where $(\underline{s}, \underline{t}) \in A$

4.1 Dichotomy search

We now discuss how to solve CP. Our idea is to perform a dichotomy search on the value u, hence pruning all student-tutor connections with reward r inferior to the current value of u. Then, the problem boils down to a special set cover problem on the set of students.

Algorithm 2: Dichotomy search for CP

Result: Optimal value $\widetilde{x}$ of CP

Initialization: Rank the scores $\{r_{s,t}\}_{s\in S', t\in T_s}$ in increasing order and form the vector $\bar{r}_1, \bar{r}_2, \ldots, \bar{r}_M$, where $M = \sum_{s\in S'} |T_s|$. Set $l = 1$ and $u = M$. Define $\mathcal{P}(a) = \{(s,t), s \in S', t \in T_s : r_{s,t} \geq a\}$. Then, denote by FP($a$) the following Feasibility Problem:

$$\begin{cases} \sum_{t:(s,t)\in\mathcal{P}(a)} x_{s,t} = 1, & \forall\, s \in S' \\ \sum_{s:(s,t)\in\mathcal{P}(a)} x_{s,t} \leq N_t, & \forall\, t \in T \\ x_{s,t} \in \{0,1\}, & \forall\, (s,t) \in \mathcal{P}(a). \end{cases} \qquad \text{(FP}(a)\text{)}$$

Solve FP(r_l) via max-flow Algorithm 3. Let S^u be the set of unassigned students.

if FP(r_l) is unfeasible **then**

  Return *unfeasible* and the set of unassigned students S^u.

Solve FP(r_u) via max-flow Algorithm 3.

if FP(r_u) is feasible **then**

  Return the student-tutor assignments with maximum score: $A = \{(\underline{s}, \underline{t}) : r_{\underline{s},\underline{t}} = r_h\}$

while $|u - l| > 1$ **do**

  Set $h = \lceil \frac{l+u}{2} \rceil$.

  Solve FP(r_h) via max-flow Algorithm 3.

  if FP(r_h) is feasible **then**

    Set $l := h$.

  else

    Set $r := h$.

Return the student-tutor assignment with lowest score: $A = \{(\underline{s}, \underline{t}) : r_{\underline{s},\underline{t}} = r_l\}$.

4.2 Max-flow for determining feasibility of FP

We are only left with solving the feasibility problem FP(a), that ask whether it is possible to assign at least one tutor to all students only using the student-tutor pairs with affinity at least a.

We next show that the problem can be converted into a Max-Flow problem, that can be solved via classic max-flow algorithms (note: the output must be integer).

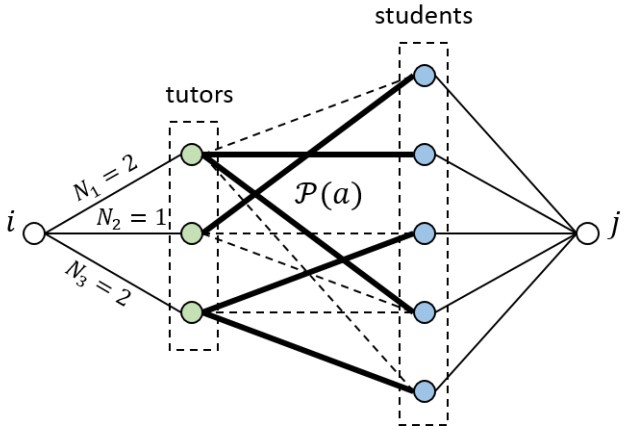

Figure 1: Illustration of Algorithm 3, solving the `max-flow` problem from i to j associated to the graph induced by the tutor-student pairs $\mathcal{P}(a)$ with affinity not smaller than a. The edges from source i to tutor t has an annotated capacity N_t. All the other edges have unitary capacity. In this case, FP(a) is feasible and the edges in bold constitute an optimal solution for FP(a).

Algorithm 3: Max-flow for FP(a)

Result: Determines whether Feasibility Problem FP(a) is feasible; if not, it returns the set of unassigned students.

1 Define the following capacitated undirected graph. Let i be the source node, connected to each tutor $t \in T$ through edge (i, t) having capacity N_t. Each tutor t is connected to student s whenever $(s, t) \in \mathcal{P}(a)$, via edge (t, s) with unitary capacity. Each student $s \in S'$ is connected to the destination node j through edge (s, j) with unitary capacity.

2 Find an integer solution to the `max-flow` problem from i to j via the Ford-Fulkerson algorithm. Let f^* be the optimal flow allocation and F^* the optimal value.

3 **if** $F^* == |S'|$ **then**

4 **Return** *feasible*, the optimal flow allocation f^* and the set of unassigned students $S^u = \emptyset$.

5 **else**

6 **Return** *unfeasible*, the optimal flow allocation f^* and the set of unassigned students $S^u = \{s : f^*_{s,j} = 0\}$.

We now show that FP(a) is a polynomial-time algorithm.

Theorem 1. *Assume that N_t is bounded for all possible instances. Then, max-flow Algorithm 3 solves the Feasibility Problem* FP(a) *in polynomial time.*

Proof. It is known that the max-flow problem can be solved by the Ford-Fulkerson algorithm in pseudo-polynomial time $\mathcal{O}(U(\#\text{nodes} + \#\text{edges}))$,

where U upper bounds the network capacity [1]. Then, if N_t is bounded by $M > 0$ then $U < M$ too, then the time is fully polynomial. Also, Ford-Fulkerson algorithm finds an integral flow whenever the capacities are integers. Then, we are left to prove that Algorithm 3 solves FP(a). In other words, we need to show that the solution of the max-flow problem in Algorithm 3 outputs value $|S'|$ if and only if FP(a) is feasible.

Assume first that FP(a) is feasible and let us denote $\hat{x}$ a feasible solution. Then, assign a unitary flow $f_{t,s}$ on all edges (t, s) such that $\hat{x}_{s,t} = 1$. Assign flow $f_{i,t} = \sum_{s:(s,t)\in\mathcal{P}(a)} \hat{x}_{s,t} \leq N_t$ to the edge (i, t) (of capacity N_t), for each $t \in T$. Finally, assign flow $f_{s,j} = 1$ to all edges of the form (s, j), for all $s \in S'$. The flow f is by construction a feasible flow from i to j in the graph defined in Algorithm 3. also, its value F equals S'. We notice that its value equals the capacity of the cut between destination j and the remaining nodes. Then, F is maximum since the value of any feasible flow is never greater than the capacity of a graph cut.

Conversely, assume that $F^* = |S'|$ is the optimal value of the max-flow problem and let f be the optimal integer flow allocation. Note first that $f_{s,j} = 1$ for all $s \in S'$. By the integral flow theorem (see [1], Thm. 6.5), since all capacities are integer (N_t or 1) then there exists an optimal integral flow allocation f. By flow conservation at node s, we have $\sum_{t:(s,t)\in\mathcal{P}(a)} f_{t,s} = f_{s,j} = 1$. Also, the flow conservation at node t and the capacity constraint on edge (i, t) impose that $\sum_{s:(s,t)\in\mathcal{P}(a)} f_{t,s} = f_{i,t} \leq N_t$. It is now easy to check that by setting $\hat{x}_{s,t} = f_{t,s}$, $\hat{x}$ is a feasible solution for FP(a). □

Corollary 1. *Iterative Max-min Algorithm 1 has polynomial-time complexity.*

4.3 ToDOs

The proposed algorithm provides the max-min allocation only if all affinities are different, i.e., $r_{s,t} \neq r_{s',t'}$ for all $s \neq s', t \neq t'$. The key is line 7 of Algorithm 2 that needs to be refined.

5 Online assignment

As time goes by, some students may depart and/or new students may subscribe to the platform, so the tutor-student assignment must be modified in a online fashion in order to deal with the new situation.

We have fundamentally two ways to deal with this online scenario. The optimal way is to recompute from scratch the overall new tutor-student assignment, by taking into account the new situation.

However, recomputing a new allocation from scratch may be too time costly, as we may need to provide a decent choice to a student that logs in for the first time. For this reason, we will devise a method to recompute the

max-min allocation on a restricted subset of students, comprising the newly subscribed one.

We finally remark that, when a new allocation is computed, the only student-tutor pair assignments that need to be maintained are those who currently have an ongoing conversation.

References

[1] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows*. Elsevier, 2014.

[2] N. Bansal and M. Sviridenko. The Santa Claus Problem. In *Proceedings of ACM Symposium on the Theory of Computation (STOC)*, page 31.

[3] Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 107–116. IEEE, 2009.

[4] Dritan Nace and Michal Pióro. Max-min fairness and its applications to routing and load-balancing in communication networks: a tutorial. *IEEE Communications Surveys & Tutorials*, 10(4), 2008.