# Iterative LexMax fair tutor-student assignment

MegaMath4SmallBiz, Frateli

May 9, 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 $a_{s,t}$ 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 [1] 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 affinity 10 ($a_{s,t} = 10$) rather than to two tutors with affinity 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 (call it *max-sum*) over all students, measured as the sum of affinities with the assigned tutors, 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 in order of decreasing affinity $a_{s_1,t} \geq a_{s_2,t} \geq \dots$ 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.

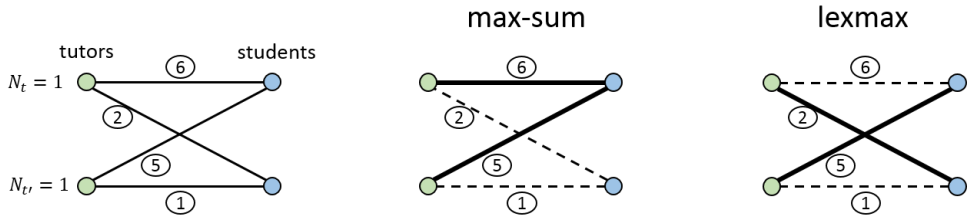


Figure 1: Comparison between max-sum and lexmax allocation. Max-sum maximizes the sum of affinities over all student-tutor pairs (= 11 here) while lexmax finds iteratively the max-min allocation.

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** (Lexicographic ordering)**.** *Let $A \in \mathbb{R}^n$ be a real vector. Let $\overline{A}$ denote $A$ sorted in increasing order. We say that $A \succ A'$ if $\overline{A}$ is lexicographically greater than $\overline{A}'$, i.e., $\exists\, n$ such that $\overline{A}_i = \overline{A}'_i$ for $i = 1, \dots, n$ and $\overline{A}_{n+1} > \overline{A}'_{n+1}$. If $A \succ A'$ or $A = A'$ we write $A \succeq A'$.*

**Definition 2** (Max-min fair allocation)**.** *[4] Let $\mathcal{F}$ be the set of feasible allocations. The max-min fair allocation $A^*$ is the lexicographic maximum among all allocations in $\mathcal{F}$, and we write $A^* = \operatorname{lex\,max}_{A \in \mathcal{F}} A$. In other words,*

$$A^* \succeq A, \quad \forall\, A \in \mathcal{F}.$$

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. For a good survey on max-min fairness we refer to [4].

Before showing how to apply the lexmax concept to our problem, let us show first a general-purpose algorithm to compute[1] the lexmax allocation.

[1]We remark that this should still be regarded as a heuristic. In order to compute the exact lexmax, one should also find among all solutions to the `REFINE` step the one that ensures that lexmax is preserved in the subsequent steps. However, this would complicate further the algorithm, hence for the time being we will consider this version.

**Algorithm 1:** General-purpose LEXMAX algorithm

**Data**: Feasible set $\mathcal{F}$
**Result**: Max-min fair allocation $a^*$
**while** $|S| > 0$ **do**
  MAX-MIN: Compute $y = \max_{A \in \mathcal{F}} \min_{k \in S} A_k$.
  REFINE: Find the allocation $A^*$ with minimum number of elements equal to $y$. Let $U = \{s : A^*_s = y\}$.
  Set $A^*_s = y$ for all $s \in U$.
  UPDATE: Update $\mathcal{F} := \{A \in \mathcal{F} : A_s = y, \ \forall\, s \in U\}$.
  Update $S := S \setminus U$.

# 4 Iterative LexMax assignment

We now describe our approach to fairly assign tutors to students, according to the concept of max-min fairness described in the previous section.

The most natural idea would be to apply the concept of max-min fairness to the vector $A(x)$, whose element $s$ measures the sum of affinities student $s$ and the assigned tutors, i.e., $A_s(x) = \sum_t x_{s,t}\, a_{s,t}$. In other words, one would solve:

$$\operatorname*{lex\,max}_{x \in \mathcal{F}} A(x)$$

where $\mathcal{F}$ is the feasibility set $\{x \in \{0,1\} : \sum_{s \in S} x_{s,t} \leq N_t, \ \forall\, t \in T\}$. On the other hand, this would lead to a solution that does not well adapt to desiderata 2, since the linearity of $A(x)$ does not incorporate the fact that students prefer to be assigned with few tutors with high affinity rather than several tutors with low affinity.

Therefore, we propose an iterative algorithm that assigns at most *one* tutor to each student at each iteration in a lexmax fashion, as long as the constraints on the maximum number of students supervised by the tutors are fulfilled. Namely, at each iteration a problem of this kind has to be solved:

$$\begin{aligned} &\operatorname*{lex\,max}_{x \in \mathcal{F}} A(x) \\ \text{s.t.} &\sum_{t \in T} x_{s,t} \leq 1, \quad \forall\, s \in S. \end{aligned}$$

We remark that 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 2.
The formal description of our procedure can be found in Algorithm 2 below.

**Algorithm 2:** Iterative LexMax fair tutor-student assignment

**Data**: Students $S$, tutors $T$, student-tutor affinity $a_{s,t}$, set of tutors $T_s = \{t : a_{s,t} > 0\}$ available to student $s$

**Result**: Max-min fair student-tutor assignment $\tau$. ($\tau_k(s) \subset T$ is the set of tutors assigned to student $s$ at iteration $k$)

**Initialization**: Set $S'' := \emptyset$ (unassigned students) and $k := 0$.

**Procedure**:

**while** $(\exists\, t :\ N_t > 0) \wedge (\exists\, s :\ |T_s| > 0)$ **do**

  Set $S' := S \setminus S''$.

  Increment iteration count $k := k + 1$.

  LEXMAX: *Allocate* one *tutor to each student in lexmax fashion*:

  **while** $(S' \neq \emptyset) \wedge (\exists\, t :\ N_t > 0)$ **do**

    MAX-MIN: Solve the max-min problem (MM) via Alg. 3:

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

    **if** $\widetilde{y}_k = 0$ **then**

      Let $S^u$ be the set of unassigned students. Set $S' := S' \setminus S^u$ and $S'' := S'' \cup S^u$. Decrement $k := k - 1$.

    **else**

      REFINE: *Find among all allocations solving* (MM) *the one with minimum number of pairs with affinity* $= y_k$:
      Let $E(y_k) \subset \mathcal{P}(y_k)$ be the set of student-tutor pairs with affinity equal to $y_k$, computed via (MM). Then, solve the Min-Cost Flow problem (MCF):

$$\begin{cases} x^* = \arg\min_x \sum_{(s,t) \in E(y_k)} x_{s,t} \\ \text{s.t. } \sum_{t:(s,t) \in \mathcal{P}(y_k)} x_{s,t} = 1, \qquad \forall\, s \in S' \\ \quad \sum_{s:(s,t) \in \mathcal{P}(y_k)} x_{s,t} \leq N_t, \qquad \forall\, t \in T \\ \quad x_{s,t} \in \{0,1\}, \qquad \forall\, (s,t) \in \mathcal{P}(y_k). \end{cases} \tag{MCF}$$

      UPDATE: Assign the tutors to the students accordingly: $\tau_k(t) = s, \forall\, (s,t) : x^*_{s,t} = 1$.

      Update the set of students yet to be assigned to a tutor: $S' := S' \setminus \{s : \exists\, t : x^*_{s,t} = 1\}$.

      Update the number of available slots for each tutor $t$: $N_t := N_t - \sum_s x^*_{s,t}$.

      Update the set of available tutors for each student: $T_s := T_s \setminus \{t : x^*_{s,t} = 1\}$

### 4.1 Dichotomy search for Max-Min problem (MM)

We now discuss how to solve the max-min problem (MM) at the generic iteration $k$, that asks to compute the maximum affinity $y_k$ such that all students are supervised by one tutor with affinity at least $y_k$.

We perform a dichotomy search on all possible $M$ affinity values, considering only the student-tutor connections $\mathcal{P}(\alpha)$ with affinity not smaller to the current affinity value $\alpha$ and solving an appropriate Max-Flow problem MF($\alpha$), as discussed in Section 4.2. Then, each student can be supervised by a tutor with affinity $\geq \alpha$ if and only if the max-flow optimal value $f(\alpha) = |S'|$.

**Algorithm 3:** Dichotomy search for Max-Min problem (MM)

**Result**: Max-min value $\widetilde{y}_k$ of (MM).

**Initialization**: Rank the scores $\{a_{s,t}\}_{s\in S', t\in T_s}$ in increasing order, eliminate the values appearing more than once and form the vector $\alpha_1, \alpha_2, \ldots, \alpha_M$, where $M \leq \sum_{s\in S'} |T_s|$. Set $l = 1$ and $u = M$. Define $\mathcal{P}(\alpha) = \{(s,t), s \in S', t \in T_s : a_{s,t} \geq \alpha\}$. Then, denote by MF($\alpha$) the following Max-Flow problem:

$$\begin{cases} f(\alpha) = \max_x \sum_{(s,t)\in\mathcal{P}(\alpha)} x_{s,t} \\ \sum_{t:(s,t)\in\mathcal{P}(\alpha)} x_{s,t} \leq 1, \qquad \forall\, s \in S' \\ \sum_{s:(s,t)\in\mathcal{P}(\alpha)} x_{s,t} \leq N_t, \qquad \forall\, t \in T \\ x_{s,t} \in \{0,1\}, \qquad \forall\, (s,t) \in \mathcal{P}(\alpha) \end{cases} \tag{MF($\alpha$))}$$

**Procedure**:

Solve MF($\alpha_l$) (see Section 4.2). Let $S^u = \{s : \sum_t x_{s,t} = 0\}$ be the set of unassigned students

**if** $f(\alpha_l) < |S'|$ **then**

  **Return** the max-min optimal value $\widetilde{y}_k := 0$ and $S^u$.

Solve MF($\alpha_u$)

**if** $f(\alpha_u) = |S'|$ **then**

  **Return** the max-min optimal value $\widetilde{y}_k := \alpha_h$ and $S^u = \emptyset$.

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

  Solve MF($\alpha_m$), where $m = \lceil \frac{l+u}{2} \rceil$

  **if** $f(\alpha_m) = |S'|$ **then**

    Set $l := m$

  **else**

    Set $r := m$

**Return** the max-min optimal value $y_k := \alpha_l$ and $S^u = \emptyset$.

We remark that the dichotomy search needs $\mathcal{O}(\log_2 M)$ iterations, therefore the Iterative Max-min Algorithm 2 has complexity $\mathcal{O}(M \log_2 M)$, where

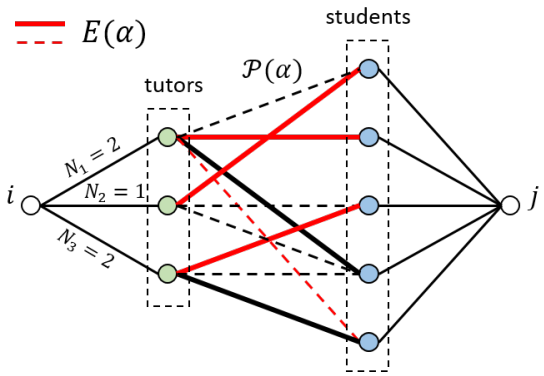


Figure 2: Illustration of Max-Flow (MF) problem from $i$ to $j$ associated to the graph induced by the tutor-student pairs $\mathcal{P}(\alpha)$ with affinity not smaller than $a$. $E(\alpha) \subset \mathcal{P}(\alpha)$ is the set of edges having affinity exactly equal to $\alpha$. The edges from source $i$ to tutor $t$ has an annotated capacity $N_t$. All the other edges have unitary capacity. In this case, the optimal value is $|S'| = 5$ hence each student is assigned to a tutor. The edges in bold constitute an optimal solution for MF($\alpha$).

$M$ is the number of different affinity values.

One possibility to reduce the algorithm complexity is is to aggregate the affinity values to lower $M$.

### 4.2 Max-Flow (MF)

We now explain how the problem (MF) can be recast as a classic max-flow one, that answers in *polynomial time* to the following question: "*Given a capacitated graph and two nodes $v, v'$, what is the maximum amount of flow that can be carried from $v$ to $v'$?*". We recall that (MF) computes the maximum number of students that can be supervised by one tutor, with student-tutor pairs with affinity at least $a$.

We define the following capacitated undirected graph (see also Fig. 2). 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.
Then, we can find an integer solution to the max-flow problem from $i$ to $j$ via, e.g., the Ford-Fulkerson algorithm. Therefore, each student can be supervised by a tutor with affinity not smaller than $\alpha$ if and only if the max-flow $f(\alpha)$ equals $|S'|$ in (MM).

### 4.3 Min-Cost Flow (MCF)

We now finally discuss how to solve line 10 of our main Algorithm 2.

We suppose we have already solved (MM) and found the maximum affinity $\widetilde{y}_k$ such that all students get one tutor each with affinity at least $y_k$. Now,

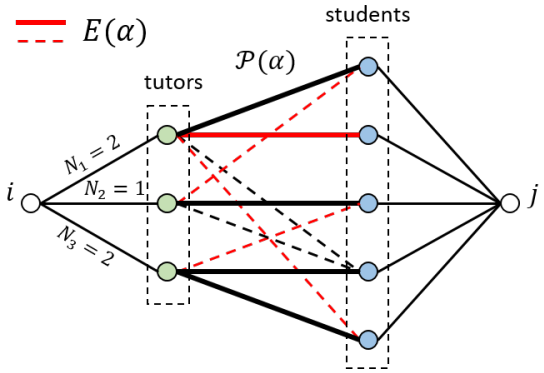


Figure 3: (continued from Fig. 2). Solving (MCF) allows to minimize the n. of pairs with affinity equal to $\alpha$ (= 1 in this case, = 3 in Fig. 2) while still assigning one tutor to each student. The edges in bald constitute the optimal tutor-student assignment; dashed edges denote all other tutor-student pairs; red edges are the ones with minimum affinity $y_k$, that one wants to avoid.

we need to find the allocation $x^*$ with the minimum number of tutor-student pairs with affinity equal to $\widetilde{y}_k$.

This problem can be reformulated via a Min-Cost flow problem (see [2], Ch. 9) or equivalently as a Transportation Problem (see [2], Ch. 10), since the tutor-student graph is bipartite.

We remark that, as in the case of Max-Flow, there exist efficient *polynomial* algorithms for solving such combinatorial problems [2].

In analogy to the previous section, let us describe the Min-Cost flow formulation. We first construct the same capacitated graph as in Section 4.2. Moreover, we append a cost to each edge, being equal to 1 if the corresponding tutor-student pair has affinity equal to $y_k$ (i.e., $E(y_k)$) and 0 otherwise. Then, the Min-Cost formulation simply asks to find the (maximum) flow of value $|S'|$ from $i$ to $j$ with minimum total cost (see Fig. 3), while fulfilling the capacity constraints on the edges. In such a way, the number of tutor-student pairs with minimum affinity $y_k$ is minimized and the lexicographic maximization is sought.

## 5 Large scale scenario

Even though the complexity of the iterative max-min Algorithm 2 is polynomial (hence, tractable) in the input size, we may need to resort to heuristics to simplify the problem in the case of large scale scenarios (large number of student $|S|$, tutors $|T|$ and affinity values $M$).

One possibility is to split the problem into several smaller sub-problems that can then be solved in parallel.

The idea is to partition the set of tutors $T$ into $n$ subsets $T_1, \ldots, T_n$ and, similarly, the set of students $S$ into $S_1, \ldots, S_n$ and solve $n$ max-min allocation problems in parallel over each "cluster" $(T_i, S_i)$. Clearly, this procedure is

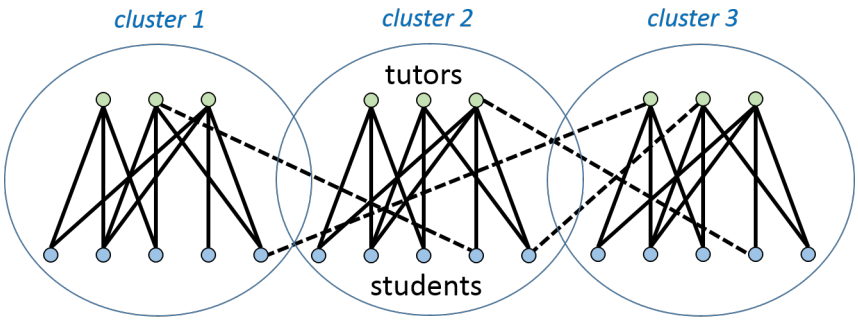


Figure 4: Tutor-student clustering. Iterative max-min Algorithm 2 is run in parallel over different clusters. This is equivalent to setting to zero all inter-cluster affinity values, corresponding to dashed edges.

optimal only if the problem is fully decoupled over different clusters, i.e., the affinity between students and tutors belonging to different clusters is null. Hence, in order to get a good approximation of the original problem we should cluster the graph such that the inter-cluster affinity is low (see Fig. 4). On the other hand, we should also ensure the clusters to have comparable size, in order to well balance the load among different processors. This can be achieved, e.g., via the bipartite graph clustering technique in [6].

Once having found an appropriate clustering, the iterative max-min Algorithm 2 is ran in parallel over each (small) cluster $(T_i, S_i)$. Notice that this is equivalent to setting to zero all inter-cluster affinities $a_{s,t}$ with $s \in S_i, t \in T_j$ with $i \neq 0$.

## 6 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 an 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 suggest to exploit the clustering idea described in Section 5. Each time a new student or tutor appears,

- we decide which cluster $(S_i, T_i)$ she will be attached to
- we recompute the iterative max-min solution *only* over the selected cluster

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

[2] Mokhtar S Bazaraa, John J Jarvis, and Hanif D Sherali. *Linear programming and network flows*. John Wiley & Sons, 2011.

[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.

[5] Stephen J Wright. *Primal-dual interior-point methods*. Siam, 1997.

[6] Hongyuan Zha, Xiaofeng He, Chris Ding, Horst Simon, and Ming Gu. Bipartite graph partitioning and data clustering. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 25–32. ACM, 2001.