Sofia University St. Kliment Ohridski
Faculty of Mathematics and Informatics
Department of Mathematical Logic and Its Applications

Master Thesis

The College Admissions problem with quotas transfer

Stefan Plamenov Fotev
Logic and Algorithms (Mathematics)
Faculty number: 25310
Sofia, 2019

Advisor:
Chief assistant, PhD Stefan Gerdjikov
Supervisor:
Chief assistant, PhD Georgi Georgiev
Department Chair:
Associate professor, PhD Hristo Ganchev

The College Admissions problem with quotas transfer

Stefan Fotev

May 16, 2019

Abstract

A different approach to the already classical College Admissions (Hospitals/Residents) problem is described. The approach is motivated by the current students ranking system used in Sofia University. It often happens that some programs are more desirable than others and this results in a certain number of unused quotas. It is natural to consider transferring such quotas to other programs that are in some sense similar to the less desired programs.

What is presented is an extension of the conditions defining a stable ranking with respect to quotas transfer rules and applicant-oriented algorithm for finding such ranking.

1 Introduction and history

The *College Admissions (Hospitals/Residents)* problem was first introduced by Gale and Shapley [1]. The name "(Hospitals/Residents)" comes from another text about this topic by Roth [2]. These are the initial papers where the so called stable ranking is defined and proven existence of. They further provide efficient algorithms to find applicant-oriented and college-oriented rankings while examining both the socio-economical and the purely theoretical aspects of the problem.

Since then such matching schemes have been applied in many countries to rank not only students but also other professionals. Some notable papers that provide background results for this and other researches are by Gusfield and Irving [3] and Roth and Sotomayor [4].

A paper that is focused on the theoretical properties of the stable rankings and providing a formal criteria for a practical selection of such ranking is for instance by Baïoua and Balinski [5].

Another interesting paper in the topic is by Biró, Fleiner, Irving and Manlove [6]. It is focused on instances of the problem where some programs have a minimal number of students that must be assigned to or several programs have restrictions on the total number of assigned students. It turns out that in general the existence of stable rankings in these cases is an NP-complete problem.

1.1 Definition of the problem

The definition of the problem as presented here includes two finite sets of objects - a set of *students* $S = \{s_1, ..., s_n\}$ and a set of *programs* $P = \{p_1, ..., p_m\}$.

There is a *quotas* function $q : P \to \mathbb{N}$, that indicates the maximum number of students that can be assigned to the corresponding program. We will refer $q(p)$ as *the quotas for program p.*

There is a *wishes* function, defined as

$$\lambda : S \to \bigcup_{k=1}^{\infty} (P \times \mathbb{Z})^k$$

and having the property that all left sides of the components of $\lambda(s)$ are unique within that tuple for all s. We will refer $\lambda(s)$ as *the list of desired programs for student s* or just *the list of s*. We will denote the total number of such wishes or $\sum_{s \in S} length(\lambda(s))$ with D_λ.

Each *wish* consists of ordered pair of a program and a *result* of the corresponding student for that program. Informally this is the criteria by which a candidate can be assigned instead of another candidate. This criteria is usually based on a predefined total order among the relevant candidates for all programs. In our terms this is equivalent with assuming that all such results for a given program are unique. However we do not assume such restrictions and will allow ties between candidates, hence considering more general problem.

The index of a program in a list indicates the preferences of the corresponding candidate, so that more desirable programs are at lower indexes than less desirable programs. The assumption that these programs are unique within each list implies that the candidate preferences for programs are indeed totally ordered, in contrast with the program preferences for candidates.

Definition 1: Given an instance $I = \langle S, P, q, \lambda \rangle$, a *stable ranking* for I is a partial function $F : S \to P$, satisfying the following conditions:

- If $F(s) = p$, then p exists in the list of s and for all programs r with lower index in that list: $|F^{-1}(r)| \geq q(r)$ and the result of s for r is less than $min(r)$
- If $F(s)$ is undefined, then for all programs p in the list of s: $|F^{-1}(p)| \geq q(p)$ and the result of s for p is less than $min(p)$
- If $|F^{-1}(p)| > q(p)$, then $min(p) =$ the $q(p)$-th greatest result for p among $F^{-1}(p)$

Here $F^{-1}(p)$ is the set of all students s, such that $F(s) = p$ and $min(p)$ is the minimal result for p among $F^{-1}(p)$ or ∞ in case this set is empty.

Also in the context of F we will refer $F^{-1}(p)$ as *the ranking for program p.*

The first two conditions define the *stability* (or fairness) property, hence the name stable ranking. These conditions say that if someone is assigned to a particular program, they have no chance of being assigned to any of their more desirable programs because these programs are full and the result of the student for any of them is not sufficient. The same holds if the student is not ranked at all.

The third condition is only affecting ties between equally preferable candidates. It says that the number of assigned students can exceed the quotas only if all exceeding candidates have the same result as the last candidate that is within the quotas (note that "exceed" and "last" are with respect to the program preferences). In such cases we will say that all candidates that share this minimal result are *trailing.*

Problem 1: Given an instance $I = \langle S, P, q, \lambda \rangle$, does there exist a stable ranking for I?

We now consider an example instance of this problem:

$$S = \{Albert, Jane, Peter\}$$
$$P = \{History, Physics\}$$
$$q = \{\langle History, 1\rangle, \langle Physics, 1\rangle\}$$
$$\lambda = \{\langle Albert, [(History, 4), (Physcis, 10)]\rangle, \langle Jane, [(Physics, 4), (History, 10)]\rangle,$$
$$\langle Peter, [(History, 4)]\rangle\}$$

A stable ranking for $I = \langle S, P, q, \lambda \rangle$ is the following function:

$$F = \{\langle Albert, History \rangle, \langle Jane, Physics \rangle, \langle Peter, History \rangle\}$$

As we see all stable ranking conditions are satisfied. Note that *Albert* and *Peter* are both assigned to $History$, although this program has only one quota. This is an example of trailing candidates.

However this stable ranking is not unique. Consider the function:

$$G = \{\langle Albert, Physics \rangle, \langle Jane, History \rangle\}$$

G also satisfies the conditions, although it is completely different from F. This small example shows that the provided conditions are not sufficient to uniquely determine a stable ranking. If we examine F and G we will see two opposing ranking tendencies. In F the candidates were ranked on their most desired program, while in G they were ranked in such way that programs assigned the best candidates.

A stable ranking is called *applicant-oriented* if it follows the first tendency and *college-oriented* if it follows the second tendency. This definition is strictly informal. Also there might be rankings that follow one of the tendencies for some programs and the other for other programs.

However there are two ranking strategies that either follow the first or the second tendency over all programs. In practical examples they often construct the same ranking.

1.2 Algorithm constructing applicant-oriented ranking

Algorithm 1 Applicant-oriented ranking

```
1:  procedure RANKING(S, P, q, λ)
2:      Set all students to Unfinalized and Non-ranked
3:      Let c(p) denote the number of students ranked in program p
4:      Let μ(p) denote the minimal result of a student ranked in program p
5:      while there exists an Unfinalized student s do
6:          Let p be the first program in the list of s for which s has not applied yet
7:          if p is undefined then
8:              Set s to Finalized
9:              continue
10:         Now s applies for p
11:         if c(p) < q(p) or the result of s for p is ≥ μ(p) then
12:             Rank s to p
13:             if c(p) > q(p) and μ(p) < the q(p)-th greatest result in p then
14:                 Set all students in p sharing μ(p) to Unfinalized and Non-ranked
15:             Set s to Finalized
```

In terms of the stable ranking function F "*Rank s to p*" means set F to be $F \setminus \{\langle x, y \rangle \in F \mid x = s\} \cup \{\langle s, p \rangle\}$ and "*Set s to Non-ranked*" means set F to be $F \setminus \{\langle x, y \rangle \in F \mid x = s\}$.

If we start with $F = \emptyset$ and consider the above remarks at the end of the while-loop F will be the desired stable matching function.

Why is this procedure correct?

The key observation is that once the quotas for a particular program are full, from then on the current minimal result for it cannot decrease. If someone is declined it means that quotas are full and their result is currently not sufficient for ranking, so by the previous statement their result will further be not sufficient. Thus we don't need to consider this application anymore (line 10). When someone is accepted it may happen that the minimal result rises so that candidates that were previously trailing are not trailing anymore and they must be declined (lines 13 and 14). If all desired programs decline a candidate, then they are non-ranked with no chance of progress and hence finalized (line 8).

It follows that whenever the procedure reaches line 5, the currently constructed function F satisfies the stable ranking conditions restricted to currently finalized candidates and everyone who is unfinalized has no chance of assignation to any program they have already applied for. It also follows that at some point all candidates will be finalized which completes the correctness.

To see that this algorithm is applicant-oriented note that every candidate is tried to assign with respect to their order of desired programs.

What is the running time of this procedure?

Let Q be the largest quotas number among all programs.

An efficient implementation includes storing the ranking for program p in multi-map data structure with results being the keys and students being the values, for instance AVL tree mapping result to list of students, sharing this result. We keep track of μ - the lowest key and m - the number of elements with that key. We delete all values with key μ when all elements in the structure become $\geq q(p) + m$. This indicates an increase in the minimal result while there are candidates exceeding the quota (maybe they were previously trailing), so they should be removed and reconsidered. Such structure can achieve $O(\log(min\{D_\lambda, Q\}))$ running time for any single insertion, deletion and updating μ and m operation, since we never search if any key-value pair already exists in the map. It is also easy to see that every student can be ranked no more than once and removed no more than once for any of their desired programs.

This guarantees $O(D_\lambda \log D_\lambda)$ running time of the procedure.

1.3 Algorithm constructing college-oriented ranking

Algorithm 2 College-oriented ranking

```
1: procedure RANKING(S, P, q, λ)
2:     Set all programs to Unfinalized
3:     Set all students to Non-ranked
4:     while there exists an Unfinalized program p do
5:         With respect to q(p) and trailing candidates take the most preferable students who:
6:             1) have p in their list and are non-ranked
7:             2) have p in their list and are ranked at less desirable program
8:         Set all programs considered at point 2) to Unfinalized
9:         Rank all taken students to p and set p to Finalized
```

Let T be a set of taken students. In terms of the stable ranking function F "*Rank all taken students to p*" means set F to be $F \setminus \{\langle s, y\rangle \in F \mid s \in T\} \cup (T \times \{p\})$.

If we start with $F = \emptyset$ and consider the above remark at the end of the while-loop F will be the desired stable matching function.

Why is this procedure correct?

The key observation is that once someone is ranked at a particular program, from then on they can only move to programs they prefer more than their current one. Thus if someone is moved because of line 7, next time we consider their old program, we will not take this candidate. Since the set of possible candidates cannot expand at some point the ranking for any program will either stabilize or become $\emptyset$. In both cases there is a point after which the program will remain finalized and so at some point all programs will be finalized.

Now if student s is not assigned correctly by first or second condition because of program p, then they were not chosen properly on line 5 when p was eventually finalized which is a contradiction. Also this selection process guarantees that the third condition is satisfied which completes the correctness.

To see that this algorithm is college-oriented note that every candidate is tried to assign with respect to the preferences of the programs.

What is the running time of this procedure?

An efficient implementation includes presorting by highest result all wishes in the corresponding programs. We keep track of the index i of the last ranked student in this sorted order and a special number q called *quotas balance*. Initially i is -1 and q is all quotas for the respective program. When we analyze a program we search for relevant candidates after i and decrease q for every accepted candidate while $q > 0$ or there are trailing candidates (remark that at some point q might become negative). We then adjust i accordingly and thus we consider every students' wish no more than once. Also whenever we detect students that are advancing by point 2) we increase the quotas balance of their previous program, indicating that they might have released a quota. We can decide if some is advancing in time $O(\log D_\lambda)$ by storing the wishes of each student in map data structure with programs being the keys and the corresponding indexes in the list being the values.

This guarantees $O(D_\lambda \log D_\lambda + |P|)$ running time of the procedure.

The running time of $O(D_\lambda \log D_\lambda)$ is not surprising. Assuming the remarks for efficient implementations both algorithms can be extended so that they report the rankings for all programs sorted by highest result for time $O(D_\lambda + |P|)$.

In the applicant-oriented algorithm we just need to report a *rightChild-root-leftChild* traversal of the multi-maps in case they indeed are implemented as AVL trees.

In the college-oriented algorithm we already have the sorted lists so we just need to report each list up the i-index while skipping candidates that are ranked in another program.

It follows that to sort an integer array $\{a_1, ..., a_n\}$ in reverse we can modify both algorithms and extend their complexity by $O(n)$ to report a sorted ranking for $I = \langle S, P, q, \lambda \rangle$ where:

$$S = \{s_1, ..., s_n\}$$
$$P = \{p\}$$
$$q = \{\langle p, n \rangle\}$$
$$\lambda = \{\langle s_1, [(p, a_1)] \rangle, ..., \langle s_n, [(p, a_n)] \rangle\}$$

Thus $O(D_\lambda \log D_\lambda)$ is a tight upper bound for finding a stable ranking.

We can now state the following theorem, which we just presented the proof of:

Theorem 1: Given an instance $I = \langle S, P, q, \lambda \rangle$, there exists a stable ranking for I. Also there exist both applicant-oriented and college-oriented algorithms to find such ranking and report the corresponding program rankings sorted by highest result for time $O(D_\lambda \log D_\lambda + |P|)$. In the general case there are no faster algorithms with the same properties.

2 Quotas transfer between pairs of programs

For a practically arising instance of the College admissions problem it may happen that there are programs that have less applications than quotas. It may also happen that there are programs sharing a common set of candidates, while one of them is significantly more desirable than the others. Both cases lead to a ranking in which there are programs with unused quotas. It is reasonable to consider transferring these quotas to other, more desired programs. In terms of ranking tendencies this process is applicant-oriented. Increasing the quotas usually leads to higher number of applicants assigned to better programs in their lists, which further leads to overall decrease in the minimal results needed for assignation.

Transferring unused quotas can also have purely social or economical benefits, both in terms of university income and student opportunities. This indeed happens in the application process for Sofia University.

We now consider a simple extension of the standard College admissions problems that allows transfer of quotas between pairs of programs.

2.1 Definition of the problem

Along with students S, programs P, quotas function q and wishes function λ, we have a *pairing table* $\Pi \subset \mathcal{P}(P)$, such that every set in Π has exactly two elements and every program belongs to exactly one element of Π.

Informally Π indicates pairs of programs that can transfer unused quotas between each other. We will call such programs *dual* and denote *the dual program of* p by ∂p, so $\{p, \partial p\} \in \Pi$ and $\partial \partial p = p$.

Often in practice p and ∂p are exactly the same university program, though not the same program during the ranking process. This separation can be based on different reasons. For instance male-female candidates, regular-part time education form, state-paid quotas and other.

Definition 2: Given an instance $J = \langle S, P, q, \lambda, \Pi \rangle$, a *stable ranking* for J is a partial function $F : S \to P$, satisfying the following conditions:

- If $F(s) = p$, then p exists in the list of s and for all programs r with lower index in that list: $|F^{-1}(r)| \geq q(r) + \max\{0,\, q(\partial r) - |F^{-1}(\partial r)|\}$ and the result of s for r is less than $min(r)$
- If $F(s)$ is undefined, then for all programs p in the list of s: $|F^{-1}(p)| \geq q(p) + \max\{0,\, q(\partial p) - |F^{-1}(\partial p)|\}$ and the result of s for p is less than $min(p)$
- If $|F^{-1}(p)| > q(p) + \max\{0,\, q(\partial p) - |F^{-1}(\partial p)|\}$, then $min(p) =$ the $q(p) + \max\{0,\, q(\partial p) - |F^{-1}(\partial p)|\}$-th greatest result for p among $F^{-1}(p)$

Again $F^{-1}(p)$ is the set of all students s, such that $F(s) = p$ and $min(p)$ is the minimal result for p among $F^{-1}(p)$ or ∞ in case this set is empty.

We now have less strict conditions for someone to be assigned, thus more strict to be declined, since not only the corresponding program must be full, but also its dual should not be able to transfer a quota. Also ties are now treated in a less strict manner, since trailing candidates are considered after the number of transferred quotas ($\max\{0,\ q(\partial p) - |F^{-1}(\partial p)|\}$) is taken in mind.

Problem 2: Given an instance $J = \langle S, P, q, \lambda, \Pi \rangle$, does there exist a stable ranking for J?

According to the definition of Π, every program must have a dual one. For each program p that we don't want to transfer quotas with, we can introduce a new dummy program p' with quota 0 and consider p and p' as dual. In the same way we can reduce an arbitrary instance of the standard problem to an instance of this problem, which makes the second more general.

2.2 Algorithm constructing applicant-oriented ranking

Algorithm 3 Applicant-oriented ranking with quotas transfer between pairs of programs

```
1: procedure RANKING(S, P, q, λ, Π)
2:     Set all students to Unfinalized and Non-ranked
3:     Let c(p) denote the number of students ranked in program p
4:     Let t(p) denote the number of quotas transferred by program p
5:     Let μ(p) denote the minimal result of a student ranked in program p
6:     while there exists an Unfinalized student s do
7:         Let p be the first program in the list of s, for which s has not applied yet
8:         if p is undefined then
9:             Set s to Finalized
10:            continue
11:        Now s applies for p
12:        if c(p) < q(p) then                          ▷ there are still unused quotas
13:            Rank s to p
14:            if c(p) + t(p) > q(p) then               ▷ we must take a quota back
15:                Now p takes back a quota from ∂p
16:                Let x = q(∂p) + t(p)
17:                if c(∂p) > x and μ(∂p) < the x-th greatest result in ∂p then
18:                    Set all students in ∂p sharing μ(∂p) to Unfinalized and Non-ranked
19:            Set s to Finalized
20:        else if c(∂p) + t(∂p) < q(∂p) then           ▷ ∂p can transfer a quota
21:            Now ∂p transfers a quota to p
22:            Rank s to p
23:            Set s to Finalized
24:        else if the result of s for p is ≥ μ(p) then
25:            Rank s to p
26:            Let x = q(p) + t(∂p)
27:            if c(p) > x and μ(p) < the x-th greatest result in p then
28:                Set all students in p sharing μ(p) to Unfinalized and Non-ranked
29:            Set s to Finalized
```

Again in terms of the stable ranking function F "*Rank s to p*" means set F to be $F \setminus \{\langle x, y\rangle \in F \mid x = s\} \cup \{\langle s, p\rangle\}$, while "*Set s to Non-ranked*" means set F to be $F \setminus \{\langle x, y\rangle \in F \mid x = s\}$.

If we start with $F = \emptyset$ and consider the above remarks at the end of the while-loop F will be the desired stable matching function.

Why is this procedure correct?

The key observation is that once all quotas for a program p are used by p and ∂p cannot transfer more quotas, from then on the current minimal result for p cannot decrease. It can alter by two reasons: someone is ranked in p with result higher than the current minimum or ∂p takes back a quota. In both cases the result either stays the same (trailing candidates) or increases. If someone is declined it means that all quotas are used by this program, the dual program cannot transfer more quotas and the result of the candidate is currently not sufficient for ranking, so by the previous statement their result will further be not sufficient. Thus we don't need to consider this application anymore (line 11). If all desired programs decline a candidate, then they are non-ranked with no chance of progress and hence finalized (line 9).

All other steps are only needed to ensure that whenever the procedure reaches line 6, the currently constructed function F satisfies the stable ranking conditions restricted to currently finalized candidates and everyone who is unfinalized has no chance of assignation to any program they have already applied for. It also follows that at some point all candidates will be finalized which completes the correctness.

What is the running time of this procedure?

As in the previous problem an efficient implementation includes storing the ranking for program p in multi-map data structure and keeping track of the lowest key and the corresponding number of elements. We also keep track of $t(p)$ and $c(p)$ for all p (they are initially 0), so again we achieve $O(1)$ for making a decision and $O(\log D_\lambda)$ for updates.

Since every student can be ranked no more than once and removed no more than once for any of their desired programs we have $O(D_\lambda \log D_\lambda)$ running time for the whole procedure.

2.3 On constructing college-oriented ranking

Consider the instance $J = \langle S, P, q, \lambda, \Pi \rangle$, where:

$$S = \{Thomas, Melanie, Martin\}$$
$$P = \{Philosophy, Biology\}$$
$$q = \{\langle Philosophy, 1\rangle \langle Biology, 2\rangle\}$$
$$\lambda = \{\langle Thomas, [(Philosophy, 10)]\rangle, \langle Melanie, [(Biology, 10)]\rangle,$$
$$\langle Martin, [(Philosophy, 9), (Biology, 9)]\rangle\}$$
$$\Pi = \{\langle Philosophy, Biology\rangle\}$$

It cannot be determined which of the following stable rankings is more acceptable for the university:

$$F = \{\langle Thomas, Philosophy\rangle, \langle Melanie, Biology\rangle, \langle Martin, Philosophy\rangle\} \text{ and}$$

$$G = \{\langle Thomas, Philosophy\rangle, \langle Melanie, Biology\rangle, \langle Martin, Biology\rangle\}$$

It turns out that the conditions that we described to define a stable ranking are not sufficient to construct a purely college-oriented ranking. In some sense this is expected since as we already

discussed the quotas transfer is an applicant-oriented process. We can however add some further restrictions to λ, so that we can indeed construct such ranking.

Suppose $J = \langle S, P, q, \lambda, \Pi \rangle$ is an instance of the problem, such that no student applied for both some program and its dual. That is for all $s \in S$ for all $e \in \Pi$ $e \not\subseteq \{x \mid \langle x, y \rangle \in \lambda(s)\}$. We will denote this property with λ^*.

Algorithm 4 College-oriented ranking with quotas transfer between pairs of programs

```
1:  procedure RANKING(S, P, q, λ*, Π)
2:      Set all programs to Unfinalized
3:      Set all students to Non-ranked
4:      while there exists an Unfinalized program p do
5:          for r ∈ {p, ∂p} do
6:              With respect to q(r) take the most preferable students who:
7:                  1) have r in their list and are non-ranked
8:                  2) have r in their list and are ranked at less desirable program
9:              Set all programs considered at point 2) to Unfinalized
10:             Rank all taken students to r
11:         if exactly one of p or ∂p still has free quotas then
12:             Let r denotes that program and let k be the free quotas of r
13:             With respect to k extend the search of candidates for ∂r as in the for loop.
14:         for r ∈ {p, ∂p} do
15:             With respect to trailing candidates extend the search for r as in the for loop.
16:             Set r to Finalized
```

Why is this procedure correct?

The key observation is the same as in the algorithm without quotas transfer, namely once someone is ranked at a particular program, from then on they can only move to programs they prefer more than their current one. To see that note that the rankings of p and ∂p for all p within the quotas are done before considering transferred quotas. Thus it cannot happen that someone is ranked in p on a transferred quota and then ∂p takes the quota back as it was in the applicant oriented algorithm. We again have that at some point all programs will be finalized. Also none of p or ∂p can be set to unfinalized on line 9, because of the assumption that no student applied both for p and ∂p. This guarantees that no relevant application can be ignored during the ranking process.

Now if student s is not assigned correctly by first or second condition because of program p, then they were not chosen properly when p was eventually finalized which is a contradiction. Also this selection process guarantees that the third condition is satisfied which completes the correctness.

What is the running time of this procedure?

As in the previous problem an efficient implementation includes presorting by highest result all wishes in the corresponding programs. We again keep track of the index i of the last ranked student in this sorted order and the quotas balance q. When we analyze a program p we search for candidates after i and decrease q for every accepted candidate while $q > 0$. Then we do the same for ∂p. If exactly one of p or ∂p has positive quotas balance (that means that the list of candidates ended before the balance reached 0) we extend the search by transfering these quotas to the balance of the other program. We further extend both searches for trailing candidates and

adjust i and q accordingly. So we consider every students' wish no more than once. Whenever we detect students that are advancing by point 2) we increase the quotas balance of their previous program, just as in the standard problem.

This guarantees $O(D_{\lambda^*} \log D_{\lambda^*} + |P|)$ running time of the procedure.

The running time needed for finding a stable ranking is still $O(D_\lambda \log D_\lambda)$. We can again report the sorted rankings of all programs in $O(D_\lambda + |P|)$ and since this problem is more general than the standard we again have that the upper bound $O(D_\lambda \log D_\lambda)$ is tight. We can now state the following theorem, which we just presented the proof of:

Theorem 2: Given an instance $J = \langle S, P, q, \lambda, \Pi \rangle$, there exists a stable ranking for J. Also there exists an applicant-oriented algorithm to find such ranking and if no student applied for some program and its dual there further exists a college-oriented algorithm to find such ranking. Both algorithms can report the corresponding program rankings sorted by highest result for time $O(D_\lambda \log D_\lambda + |P|)$. In the general case there are no faster algorithms with the same properties.

3 Arbitrary quotas transfer

The previous section describes a relatively simple extension of the standard problem, where pairs of programs can transfer quotas between each other. Unfortunately transfering quotas between several programs is not so natural to define. Consider the following instance of the standard problem:

$$S = \{Anna, George, Michael, Stephanie\}$$
$$P = \{Psychology, Economics, Law\}$$
$$q = \{\langle Psychology, 1\rangle \langle Economics, 1\rangle \langle Law, 1\rangle\}$$
$$\lambda = \{\langle Anna, [(Psychology, 10)]\rangle, \langle George, [(Law, 10)]\rangle, \langle Michael, [(Psychology, 9)]\rangle,$$
$$\langle Stephanie, [(Law, 9)]\rangle\}$$

If we allow quotas transfer in directions $Economics \to Psychology$ and $Economics \to Law$ we can obtain two different rankings:

$$F = \{\langle Anna, Psychology\rangle, \langle George, Law\rangle, \langle Michael, Psychology\rangle\} \text{ and}$$

$$G = \{\langle Anna, Psychology\rangle, \langle George, Law\rangle, \langle Stephanie, Law\rangle\}$$

In F we transfered the $Economics$ quota to $Psychology$ while in G it was transfered to Law.

In this example there occurs a tie between two programs for a single quota. In order for a ranking to be fair there must exist formal rules that define how will unused quotas be transferred in case of such ties. All candidates must be aware of these rules before the beginning of the application process because this information can significantly affect their order of desired programs.

We now introduce such rules. Suppose $P = \{p_1, ..., p_5\}$ is a set of programs. Consider the following directed graph:

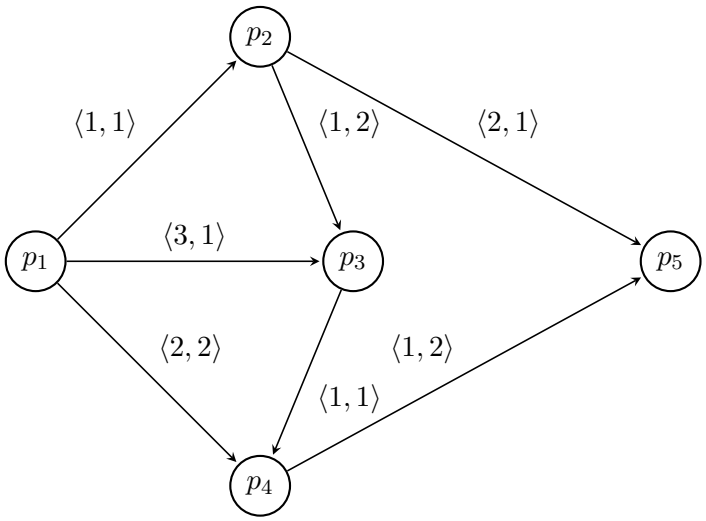


Fig.1 Simple transfer graph

Every edge has two natural number characteristics. We will call the left component *out-priority* and the right component *in-priority*. We also want every node to have consecutive, starting from 1 out-priorities for out-coming edges and consecutive, starting from 1 in-priorities for in-coming edges as shown in the figure. The are no other restrictions for this graph, except that it must be irreflexive. There might be nodes with no edges or nodes that have edges with only one type of direction (like p_1 and p_5).

The intuition behind this is the following: Suppose program p_1 has an unused quota. Then it is a priority for p_1 to transfer this quota to p_2, then to p_4 and finally to p_3, just as the corresponding out-priorities. Suppose p_5 needs a quota. Then it is a priority for p_5 to take this quota from p_2 and only then from p_4, just as the corresponding in-priorities.

3.1 Definition of the problem

Along with students S, programs P, quotas function q and wishes function λ, we have a *transfer table* $T \subset P \times P$ and *priority functions* $\sigma^{out} : T \to \mathbb{N}$ and $\sigma^{in} : T \to \mathbb{N}$, having the following properties:

- If $\langle p_{out}, p_{in}\rangle \in T$, then $p_{out} \neq p_{in}$
- If $p \in P$ and $|T^p| = x$, then $\{\sigma^{out}(e) \,|\, e \in T^p\} = \{1, ..., x\}$
- If $p \in P$ and $|T_p| = y$, then $\{\sigma^{in}(e) \,|\, e \in T_p\} = \{1, ..., y\}$

Here T^p denotes $\{\langle p_{out}, p_{in}\rangle \in T \,|\, p = p_{out}\}$ and T_p denotes $\{\langle p_{out}, p_{in}\rangle \in T \,|\, p = p_{in}\}$

With respect to the graph example, T is the set of edges and σ^{out} and σ^{in} are functions defining the out-priorities and in-priorities correspondingly.

Definition 3: Given T, σ^{out} and σ^{in} for $\langle p, r\rangle \in T$, we define *the set of higher out-edges* as $T^{p\uparrow r} = \{e \in T^p \,|\, \sigma^{out}(e) < \sigma^{out}(\langle p, r\rangle)\}$ and *the set of higher in-edges* as $T_{r\uparrow p} = \{e \in T_r \,|\, \sigma^{in}(e) < \sigma^{in}(\langle p, r\rangle)\}$.

In the previously described problems the stable ranking was defined solely by the ranking function F, mapping students to programs. In this problem, however, we will include an additional function $\tau : T \to \mathbb{N}$, indicating the number of transferred quotas between applicable pairs of programs. This will be relevant for the verification that the ranking is indeed fair.

Because of the newly introduced concept of out-priorities we also introduce a special characteristics $\kappa : T \to \mathbb{N}$, defined as $\kappa(\langle p, r\rangle) = \max\{0,\, q(p) - |F^{-1}(p)| - \sum_{e \in T^{p\uparrow r}} \tau(e)\}$. Informally this function indicates the maximal amount of quotas that program p can transfer to program r with respect to τ and out-priorities.

We now describe the intuition and motivation for the stable ranking conditions in this extension of the problem.

- In the standard problem we have that if a student s is ranked in program p, then all more desirable programs for s are full and their minimal result is higher than the result of s. Since we now allow quotas transfers it is natural to extend this with the additional condition that all such programs cannot find a quota for s, i.e they have collected all quotas they could collect with respect to κ. The same holds if the student is not ranked at all.
- Trailing candidates are considered in analogous manner - someone can be ranked as trailing only if the corresponding program has already collected all quotas it could collect.
- Of course a program can transfer a quota only if this quota is not needed for its own ranking and also if a program collected a quota, then its own quotas are already full.
- The last extension is regarding the concept of in-priorities namely if program p transferred quotas to program r, then r collected the maximal possible amount of quotas from all programs with in-priority higher than p.

With this intuition in mind the formal definition goes as follows:

Definition 4: Given an instance $K = \langle S, P, q, \lambda, T, \sigma^{out}, \sigma^{in}\rangle$, a partial function $F : S \to P$ and a function $\tau : T \to \mathbb{N}$, we define $\kappa : T \to \mathbb{N}$ as $\kappa(\langle p, r\rangle) = \max\{0,\, q(p) - |F^{-1}(p)| - \sum_{e \in T^{p\uparrow r}} \tau(e)\}$. We say that the ordered pair $\langle F, \tau\rangle$ is a *stable ranking* for K if it satisfies the following conditions:

1. If $F(s) = p$, then p exists in the list of s and for all programs r with lower index in that list: $|F^{-1}(r)| \geq q(r) + \sum_{e \in T_r} \kappa(e)$ and the result of s for r is less than $min(r)$

2. If $F(s)$ is undefined, then for all programs p in the list of s: $|F^{-1}(p)| \geq q(p) + \sum_{e \in T_p} \kappa(e)$ and the result of s for p is less than $min(p)$

3. If $|F^{-1}(p)| > q(p) + \sum_{e \in T_p} \tau(e)$, then $min(p) =$ the $q(p) + \sum_{e \in T_p} \tau(e)$-th greatest result for p among $F^{-1}(p)$ and for all $e \in T_p$: $\tau(e) = \kappa(e)$

4. If $\sum_{e \in T^p} \tau(e) = x > 0$, then $|F^{-1}(p)| \leq q(p) - x$

5. If $\sum_{e \in T_p} \tau(e) = x > 0$, then $|F^{-1}(p)| \geq q(p) + x$

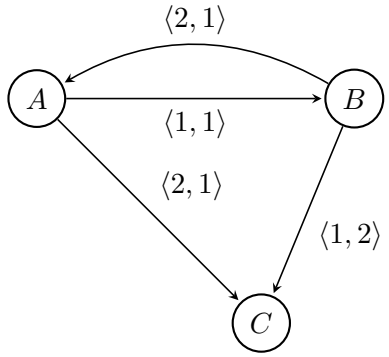
6. If $\tau(\langle p, r\rangle) > 0$, then for all $\langle u, r\rangle \in T_{r\uparrow p}$: $\kappa(\langle u, r\rangle) = \tau(\langle u, r\rangle)$

Here $F^{-1}(p)$ is the set of all students s, such that $F(s) = p$ and $min(p)$ is the minimal result for p among $F^{-1}(p)$ or ∞ in case this set is empty.

Problem 3: Given an instance $K = \langle S, P, q, \lambda, T, \sigma^{out}, \sigma^{in} \rangle$, does there exist a stable ranking for K?

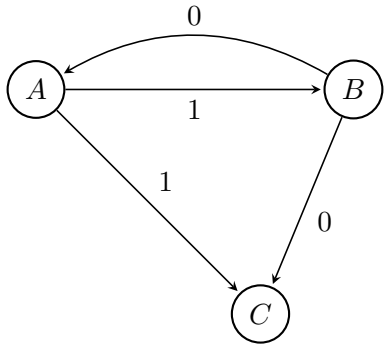
We now consider an example instance of this problem.
Suppose $L = \langle S, P, q, \lambda, T, \sigma^{out}, \sigma^{in} \rangle$, where:

$$S = \{Tim, Jane, John, Peter, Anna, Stefan, Klaudia\}$$
$$P = \{A, B, C\}$$
$$q = \{\langle A, 3\rangle, \langle B, 1\rangle, \langle C, 1\rangle\}$$
$$\lambda = \{\langle Tim, [(C, 7), (B, 10)]\rangle, \langle Jane, [(B, 8)]\rangle, \langle John, [(A, 10)]\rangle, \langle Peter, [(C, 9)]\rangle,$$
$$\langle Anna, [(C, 9)]\rangle, \langle Stefan, [(C, 9)]\rangle, \langle Klaudia, [(C, 8)]\rangle\}$$
$$T, \sigma^{out}, \sigma^{in} =$$



The following lists with results and graph with number of transfered quotas is a description of a stable ranking for L:

$$A = [John\text{-}10], \quad B = [Tim\text{-}10, Jane\text{-}8], \quad C = [Anna\text{-}9, Stefan\text{-}9, Peter\text{-}9]$$
$$\tau =$$



Anna, $Stefan$ and $Peter$ are ranked as trailing, because of condition 3.

Klaudia is not ranked, because their result is not sufficient for C. Also $\kappa(A, C) = \tau(A, C) = 1$ and $\kappa(B, C) = \tau(B, C) = 0$, so the second stable ranking condition is indeed valid for them. If they, however, had an additional wish for program B, they would have been ranked there, because it is a priority for A to transfer a quota to B. Note that this would not have changed the ranking of C.

Tim is not ranked on his top desired program, because of condition 1.

3.2 Algorithm constructing applicant-oriented ranking

We now introduce an algorithm that constructs applicant-oriented ranking by the above rules. We can summarize it as follows: we try to assign each student s with respect to their desired programs. Let p be the first not analyzed yet. Now:

- If there are quotas, not used by p we rank s in p and check if we have to take back a quota and how this affects the ranking of the corresponding program.
- If p is already full, we try to find a quota with respect to in-and-out priorities and check if this affects the rankings of other programs.
- If p cannot find a quota, we try to assign s by result and check what happens with trailing candidates.
- If the result of s is not sufficient, then it will further be not sufficient so p declines s.

Algorithm 5 Applicant-oriented ranking with arbitrary quotas transfer

procedure RANKING($S, P, q, \lambda, T, \sigma^{out}, \sigma^{in}$)
  Set all students to *Unfinalized* and *Non-ranked*
  Let $c(p)$ denote the number of students ranked in program p or $|F^{-1}(p)|$
  Let $t^{out}(p)$ denote the number of quotas transferred by program p or $\sum_{e \in T^p} \tau(e)$
  Let $t^{in}(p)$ denote the number of quotas received by program p or $\sum_{e \in T_p} \tau(e)$
  Let $\mu(p)$ denotes the minimal result of a student ranked in program p
  while there exists an *Unfinalized* student s **do**
    Let p be the first program in the list of s that still has not declined s
    if p is undefined **then**
      Set s to *Finalized*
      continue
    if $c(p) < q(p)$ **then** ▷ *there are still unused quotas*
      Rank s to p
      if $c(p) + t^{out}(p) > q(p)$ **then** ▷ *we must take a quota back*
        Let r be the last program by out-priority for which p transferred a quota
        Now p takes back a quota from r and declines r
        if all programs that r can take a quota from declined r **then**
          REBALANCE(r)
        else
          Set one student in r sharing $\mu(r)$ to *Unfinalized* and *Non-ranked*
      Set s to *Finalized*
      continue
    Let r be the first program by in-priority that still has not declined p

```
24:     while r is defined do
25:         Let foundQuota be false
26:         if c(r) + t^out(r) < q(r) then                    ▷ r has a free quota
27:             Set foundQuota to true
28:         else if t^out(r) > 0 then
29:             Let u be the last program by out-priority for which r transferred a quota
30:             if σ^out(⟨r, p⟩) < σ^out(⟨r, u⟩) then
31:                 Now r takes back a quota from u and declines u
32:                 if all programs that u can take a quota from declined u then
33:                     REBALANCE(u)
34:                 else
35:                     Set one student in u sharing μ(u) to Unfinalized and Non-ranked
36:                 Set foundQuota to true
37:         if foundQuota = true then
38:             Now r transfers a quota to p
39:             Rank s to p
40:             Set s to Finalized
41:             break
42:         else
43:             Now r declines p
44:             Set r to be the next program by in-priority for p after r
45:     if r is undefined then                                ▷ we couldn't find a quota
46:         if the result of s for p is ≥ μ(p) then
47:             Rank s to p
48:             REBALANCE(p)
49:         else
50:             Now p declines s
51: procedure REBALANCE(p)
52:     Let x = q(p) + t^in(p)
53:     if c(p) > x and μ(p) < the x-th greatest result in p then
54:         Now p declines all students in p sharing μ(p)
55:         Set all students in p sharing μ(p) to Unfinalized and Non-ranked
```

Again in terms of F "*Rank s to p*" means set F to be $F \setminus \{\langle x, y\rangle \in F \mid x = s\} \cup \{\langle s, p\rangle\}$, while "*Set s to Non-ranked*" means set F to be $F \setminus \{\langle x, y\rangle \in F \mid x = s\}$. Also "*$p$ transfers a quota to r*" means increase $\tau(\langle p, r\rangle)$ and "*p takes back a quota from r*" means decrease $\tau(\langle p, r\rangle)$.

If we start with $F = \emptyset$, $\tau = T \times \{0\}$ and consider the above remarks at the end of the while-loop $\langle F, \tau\rangle$ will be the desired stable ranking.

Why is this procedure correct?

Lemma 1: *The above procedure terminates.*

Whenever a student s is set to unfinalized in the while-loop, this is done either after they are declined by some program, because of rebalancing or after their previous program was declined by another program (lines 20 and 35). Both situations can occur only finitely many times after which s will be set to finalized by line 10 and will remain finalized. Since this holds for arbitrary s, we have that at some point all students will be finalized and the while-loop will end.

Lemma 2: *When the above procedure reaches line* 22 *and* $c(p) = q(p)$ *for currently considered program* p, $t^{out}(p) = 0$ *and from now on this value will not change.*

(1) We increase $t^{out}(p)$ only if the contidion on line 26 is fullfilled, namely $c(p)+t^{out}(p) < q(p)$.

Whenever we rank a student by line 12, we increase $c(p)$ and then evenutally decrease $t^{out}(p)$ on line 16. By this and (1) it follows that whenever the procedure reaches line 22, $c(p)+t^{out}(p) \leq q(p)$ for currently considered program p. Thus if $c(p) = q(p)$ on that line, we have that $t^{out}(p) = 0$ and this value will not change anymore again because of (1).

Corollary of Lemma 2: *During the run of the above procedure, if program* p *is declined or recieves a quota from program* r, *then* $t^{out}(p) = 0$ *and from now on this value will not change.*

Consider the first time program p is trying to find a quota for student s. It follows that currently $c(p) = q(p)$, so for the last student ranked within the quotas we have that the procedure reached line 22, which implies that $t^{out}(p) = 0$ and this value will not change.

Theorem 3 (invariant of the main while-loop): *Whenever the procedure reaches line* 7, *the currently constructed functions* F *and* τ *satisfy the stable ranking conditions restricted to currently finalized candidates, everyone who is unfinalized has no chance of assignation to any program that declined them and every program has no chance of receiving a quota from a program that declined it.*

Proof. We prove the invariant by considering different cases as described in the beginning of this sub-chapter.

Before the beginning of the while-loop there are no finalized students, so F being $\emptyset$ and τ being $T \times \{0\}$ vacuously satisfy all stable ranking conditions restricted to finalized candidates. The other two invariant conditions are also satisfied because no program declined any student or other program.

Suppose the invariant conditions are satisfied for currently constructed F and τ and we are analyzing student s.

• If all the desired programs have already declined s, then they are set to finalized. This does not change the definitions of F and τ and we already have that s has no chance of assignation by the invariant, so F and τ now satisfy the second stable ranking condition also for s.

• If the currently considered program p has quotas, not used by it (line 12), we just rank s here. However this might cause the fourth stable ranking condition to be no longer satisfied. If so, we take back a quota from the last program by out-priority r that p transferred to, noting that taking back the quota from other program might cause the failure of condition six or further failure of conditions one or two. By the very same reason we see that p can not give any more quotas to r and thus p declines r, keeping the third invariant condition. This quota may cause a student (or students) to drop from the ranking of r. We now have two cases:

1) all programs that r can receive a quota from already declined it and so we need to check only if the third stable ranking condition is still satisfied. If it's not, we know that r cannot receive any more quotas and students who are trailing now have a lower result than the last within the taken quotas. By Corollary of Lemma 2, we have that $t^{out}(r) = 0$ and this value will not change. It follows that the minimal result can now alter only if someone with higher result is ranked or r gives a quota back, which causes it to either increase or stay the same. In both cases this minimum cannot decrease and thus the dropped candidates do not have any chance of further assignation, so r declines them, keeping the second invariant condition;

2) we might find a quota for r. In this case we just drop one student sharing the minimal result and will reconsider their application later.

• We now consider the case when we have to search for a quota. By the invariant we know that all the programs that declined p already transferred to p the maximal possible number of quotas. Thus we start the search from the first program by in-priority that still has not declined p. This program r can give quotas to p only if it has unused quotas or has already given a quota to a program with lower out-priority than p. If so, we proceed with that program exactly as in the previous case. Note that this whole step is performed after we are sure that there are no free quotas in p and so this student is indeed ranked on a taken quota, causing the fifth stable ranking condition to be satisfied. Note also that it does not matter what the result of this student is. We may just consider that always the candidates with worst results are taking the transferred quotas.

• If we cannot find a quota for s we try to rank them by result and in case there are dropping candidates we proceed in a way similar to the second case, so the invariant conditions are satisfied by the same reasons. If now s cannot be ranked by result, they have no further chance of assignation and so they are declined, again keeping the invariant conditions.

□

From Lemma 1 we have that at some point all students will be finalized and the while-loop will end. From the invariant conditions we now have that F and τ satisfy the stable ranking conditions over all candidates, which completes the correctness of the procedure.

What is the running time of this procedure?

Let ρ denotes the total number of programs or $|P|$.

Suppose $e = \langle p, r\rangle \in T$. For each such r we set p at the $\sigma^{in}(e) - 1$-st position in an array for r. We start searching for quotas from the beginning of this array and continue from the next element whenever r is denied. Also for each such p we put the pairs $(r, \sigma^{out}(e))$ in a map data structure for p and r on the $\sigma^{out}(e) - 1$-st position in an array for p. For each p we need one more initially empty structure mapping out-priorities with number of transferred quotas. In this structure we keep track of the highest key.

Preparing the above mentioned structures costs $O(\rho^2 \log \rho)$.

We can now answer in time $O(\log \rho)$ for the question on line 30, since $\sigma^{out}(\langle r, u\rangle)$ is just the highest key in the structure maintaining the number of transferred quotas. It also takes $O(\log \rho)$ to update this structure.

Of course we keep track of $c(p)$, $t^{out}(p)$ and $t^{in}(p)$ for all p. We initialize them with 0.

As in the previous problems we store the ranking for program p in multi-map data structure and keep track of the lowest key and the corresponding number of elements. This again guarantees $O(1)$ time needed for making a decision whether someone should be ranked or removed and $O(\log D_\lambda)$ for actual ranking or removing.

We now observe that every candidate can be ranked and removed no more than $O(\rho)$ times to each of their desired programs. They can be ranked once in the main ranking of program p, once for all programs that p can receive a quota from and once as a trailing candidate, while p has been declined by all relevant programs. One such step costs $O(\log D_\lambda)$ to update the ranking and $O(\log \rho)$ to update the transfers.

This guarantees $O(\rho D_\lambda(\log \rho + \log D_\lambda) + \rho^2 \log \rho)$ running time for the whole procedure.

Reporting the actual program rankings and transferred quotas is now just iterating over all elements in the corresponding multi-maps and maps. The time complexity of this is $O(D_\lambda + \rho^2)$ which does not increase the overall running time of the procedure.

We can now summarize the results of this subchapter with the following theorem:

Theorem 4: Given an instance $K = \langle S, P, q, \lambda, T, \sigma^{out}, \sigma^{in} \rangle$, there exists a stable ranking for K. Also there exists an applicant-oriented algorithm to find such ranking and report the corresponding program rankings and number of transferred quotas in time $O(\rho D_\lambda(\log \rho + \log D_\lambda) + \rho^2 \log \rho)$, where ρ is the number of programs.

3.3 On constructing college-oriented ranking

Consider the instance $K = \langle S, P, q, \lambda, T, \sigma^{out}, \sigma^{in} \rangle$, where:

$$S = \{John, Alexandra\}$$
$$P = \{Chemistry, Pharmacy, Ecology\}$$
$$q = \{\langle Chemistry, 1\rangle, \langle Pharmacy, 1\rangle, \langle Ecology, 0\rangle\}$$
$$\lambda = \{\langle John, [(Ecology, 8), (Pharmacy, 10)]\rangle, \langle Alexandra, [(Pharmacy, 10)]\rangle\}$$
$$T, \sigma^{out}, \sigma^{in} = \{(Chemistry, Pharmacy, 1, 1), (Chemistry, Ecology, 2, 1)\}$$

It cannot be determined which of the following stable rankings is more acceptable for the university:

$$F_1 = \{\langle Alexandra, Pharmacy\rangle, \langle John, Ecology\rangle\},$$
$$\tau_1 = \{Chemistry \xrightarrow{0} Pharmacy, Chemistry \xrightarrow{1} Ecology\}$$
and
$$F_2 = \{\langle Alexandra, Pharmacy\rangle, \langle John, Pharmacy\rangle\},$$
$$\tau_2 = \{Chemistry \xrightarrow{1} Pharmacy, Chemistry \xrightarrow{0} Ecology\}$$

This undeterminacy is again caused by a student applying for two programs that can collect common quotas. In the previous section we solved this by adding the restriction that no student can apply for both a program and its dual. As mentioned this assumption arises from purely practical cases in which the dual programs have some conceptual separation that prevents students from applying for both (for instance male-female candidates, different education form or something else). This assumption however is not applicable here, since we expect that the transfer table is based on similarities between programs and consecutively it is expected for a candidate to apply for several similar programs.

We can conclude that the above described conditions for a stable ranking are not sufficient to decide what is a purely college-oriented ranking.

4 Further observations and conclusions

An important observation is that a candidate can be ranked on a less desirable program in their list, because of a non-ranked candidate. Thus it is reasonable to consider reporting also the non-ranked candidates.

We again consider the example $K = \langle S, P, q, \lambda, T, \sigma^{out}, \sigma^{in} \rangle$, where:

$$S = \{John, Alexandra\}$$
$$P = \{Chemistry, Pharmacy, Ecology\}$$
$$q = \{\langle Chemistry, 1\rangle, \langle Pharmacy, 1\rangle, \langle Ecology, 0\rangle\}$$
$$\lambda = \{\langle John, [(Ecology, 8), (Pharmacy, 10)]\rangle, \langle Alexandra, [(Pharmacy, 10)]\rangle\}$$
$$T, \sigma^{out}, \sigma^{in} = \{(Chemistry, Pharmacy, 1, 1), (Chemistry, Ecology, 2, 1)\}$$

The algorithm will construct the following stable ranking:

$$F_1 = \{\langle Alexandra, Pharmacy\rangle, \langle John, Ecology\rangle\},$$
$$\tau_1 = \{Chemistry \xrightarrow{0} Pharmacy, Chemistry \xrightarrow{1} Ecology\}$$

Suppose there is another candidate *Bertha* with list $[(Pharmacy, 9)]$. In this case the only stable ranking is:

$$F_2 = \{\langle Alexandra, Pharmacy\rangle, \langle John, Pharmacy\rangle\},$$
$$\tau_2 = \{Chemistry \xrightarrow{1} Pharmacy, Chemistry \xrightarrow{0} Ecology\}$$

Although *Bertha* is not ranked, this candidate is forcing *John* to drop to a less desirable program in their list. Note also that the pair $\langle F_1, \tau_1 \rangle$ is no longer stable, since it contradicts the second stable ranking condition for *Bertha*.

This new ranking opens another question regarding fairness: Why should *John* receive a quota for *Pharmacy* - can this candidate be ranked there as trailing?

For that we first need to drop the second part of the third stable ranking condition and formulate it just as: If $|F^{-1}(p)| > q(p) + \sum_{e \in T_p} \tau(e)$, then $min(p) =$ the $q(p) + \sum_{e \in T_p} \tau(e)$-th greatest result for p among $F^{-1}(p)$. If we now introduce the restriction that trailing candidates take only one quota, then a stable ranking might not exist.

Consider:

$$F_3 = \{\langle Alexandra, Pharmacy\rangle, \langle John, Pharmacy\rangle, \langle Bertha, Pharmacy\rangle\},$$
$$\tau_3 = \{Chemistry \xrightarrow{1} Pharmacy, Chemistry \xrightarrow{0} Ecology\}$$

This ranking is not stable, because it contradicts the reformulated third condition for *Pharmacy*. The pair $\langle F_2, \tau_2 \rangle$ is not stable too, because it contradicts the newly introduced rule. It is easy to check that all other possible rankings for this instance are also not stable.

We currently don't know if the existance of a stable ranking for the problem with the above mentioned restrictions is in the NP class.

Another interesting question would be to find a stable ranking for the problem with reformulated third condition, such that the trailing candidates indeed take the minimum number of transferred quotas. That is to find a stable ranking $\langle F, \tau \rangle$ for arbitrary instance $K = \langle S, P, q, \lambda, T, \sigma^{out}, \sigma^{in} \rangle$, such that $\sum_{e \in T} \tau(e)$ is minimal.

Is it possible to define the stable ranking conditions in such way, so that there exists a college-oriented ranking?

As shown in the previous section, our definition of the problem appears to be not sufficient to decide what is a purely college-oriented ranking. There however might exist other extensions, for which a college-oriented ranking is defined. Along with the tansfer graph we can introduce a linearly ordered *importance* property for the programs. Our extension is still more general, since we can easily introduce in-and-out priorities, based on this ordering (we take quotas from less important programs and give to more important). What can now be considered a college-oriented tendency is transferring quotas to programs with higher importance, but for now we leave this is as an open problem.

What would happen if we ignore the concept of in-priorites and define stable ranking without the sixth condition?

Consider the instance $M = \langle S, P, q, \lambda, T, \sigma^{out} \rangle$, where:

$$S = \{Thomas, Kim, Joanna\}$$
$$P = \{A, B, C, D, E\}$$
$$q = \{\langle A, 0\rangle, \langle B, 0\rangle, \langle C, 0\rangle, \langle D, 1\rangle, \langle E, 1\rangle\}$$
$$\lambda = \{\langle Thomas, [(A, 8)]\rangle, \langle Kim, [(B, 7)]\rangle, \langle Joanna, [(C, 8)]\rangle\}$$
$$T, \sigma^{out} = \{(D, B, 1), (D, A, 2), (E, B, 1), (E, C, 2)\}$$

It cannot be determined which of the following rankings is more fair for the candidates:

$$F_4 = \{\langle Thomas, A\rangle, \langle Kim, B\rangle\},$$
$$\tau_4 = \{D \xrightarrow{0} B, D \xrightarrow{1} A, E \xrightarrow{1} B, E \xrightarrow{0} C\}$$
and
$$F_5 = \{\langle Kim, B\rangle, \langle Joanna, C\rangle\},$$
$$\tau_5 = \{D \xrightarrow{1} B, D \xrightarrow{0} A, E \xrightarrow{0} B, E \xrightarrow{1} C\}$$

Is it possible to construct another (possibly faster) ranking method if we are allowed to break some of the stable ranking conditions? For instance, if some students do not mind in which desired program they are ranked.

Breaking any of the stable ranking conditions opens questions regarding fairness, especially if there occurs a tie between candidates. We leave this also as an open problem.

References

[1] D. Gale, L.S. Shapley, *College admissions and the stability of marriage*, American Mathematical Monthly 69 (1) (1962) 9-15.

[2] A.E. Roth, *The evolution of the labor market for medical interns and residents: a case study in game theory*, Journal of Political Economy 6 (4) (1984) 991-1016.

[3] D. Gusfield, R.W. Irving, *The Stable Marriage Problem: Structure and Algorithms*, MIT Press, 1989.

[4] A.E. Roth, M.A.O. Sotomayor, *Two-sided matching: a study in game-theoretic modeling and analysis*, Econometric Society Monographs, vol. 18, Cambridge University Press, 1990.

[5] M. Baïoua, M. Balinski *Two-sided matching: a study in game-theoretic modeling and analysis*, Theoretical Computer Science 322 (2004) 245 – 265

[6] P. Biró, T. Fleiner, R.W. Irving, D.F. Manlove, *The College Admissions problem with lower and common quotas*, Theoretical Computer Science 411 (2010) 3136-3153