

# Envy-free matchings with cost-controlled quotas

Girija Limaye and Meghana Nasre

Indian Institute of Technology Madras, India  
{girija,meghana}@cse.iitm.ac.in

**Abstract.** We consider the problem of assigning agents to programs in the presence of two-sided preferences, commonly known as the Hospital Residents problem. In the standard setting each program has a rigid upper-quota which cannot be violated. Motivated by applications where quotas are governed by resource availability, we propose and study the problem of computing optimal matchings with cost-controlled quotas – denoted as the CCQ setting.

In the CCQ setting we have a cost associated with every program which denotes the cost of matching a single agent to the program. Our goal is to compute a matching that matches *all* agents, respects the preference lists of agents and programs and is optimal with respect to the cost criteria. We consider *envy-freeness* as a notion of optimality and study two optimization problems with respect to the costs – minimize the total cost (MINSUM) and minimize the maximum cost at a program (MINMAX). We show that there is a sharp contrast in the complexity status of these two problems – MINMAX is polynomial time solvable whereas MINSUM is NP-hard and hard to approximate within a constant factor unless  $P = NP$  even under severe restrictions. On the positive side, we present approximation algorithms for the MINSUM for the general case and a special hard case. We achieve the approximation guarantee for the special case via a technically involved linear programming (LP) based algorithm. We remark that our LP is for the general case of the problem.

**Keywords:** Matchings under two-sided preferences, Envy-free matchings, Cost-based allocation, Hardness of approximation, Approximation algorithms, Linear Programming

## 1 Introduction

The problem of computing optimal matchings in the many-to-one setting and two-sided preferences has been extensively investigated since it is applicable in several real-world applications like assigning students to schools [2] or elective courses [21], under-graduate students to university programs [4], medical interns (resident doctors) to hospitals [24] and many more. This setting, commonly called as the Hospital Residents (HR) setting, is modelled as a bipartite graph  $G = (\mathcal{A} \cup \mathcal{P}, E)$  where  $\mathcal{A}$  and  $\mathcal{P}$  denote the set of agents and programs respectively and  $(a, p) \in E$  if and only if  $a \in \mathcal{A}$  and  $p \in \mathcal{P}$  are mutually acceptable to each other. For a vertex  $v$ , let  $\mathcal{N}(v)$  denote the vertices adjacent to  $v$ . Each vertex  $v \in \mathcal{A} \cup \mathcal{P}$  ranks the elements in  $\mathcal{N}(v)$  in a strict order and this ranking is called as the *preference list* of the agent or program. If  $x$  prefers  $y$  over  $z$ , we denote it by  $y \succ_x z$ . In the standard setting a program  $p$  has a positive upper-quota  $q(p)$ .

A *matching*  $M \subseteq E$  in the HR instance  $G$  is an assignment of agents to programs such that each agent is matched to at most one program and a program is matched to at most  $q(p)$  many agents. Let  $M(a)$  denote the program that agent  $a$  is matched to in  $M$  ( $M(a) = \perp$  if  $a$  is unmatched) and  $M(p)$  denote the set of agents matched to program  $p$  in matching  $M$ . An agent  $a$  prefers being matched to one of the programs in  $\mathcal{N}(a)$  over remaining unmatched. A program  $p$  is called *under-subscribed* in  $M$  if  $|M(p)| < q(p)$ . *Stability* is a well-accepted notion of optimality in this setting and is defined as follows.

**Definition 1 (Stable matchings in HR setting).** A pair  $(a, p) \in E \setminus M$  is a *blocking pair* w.r.t. the matching  $M$  if  $p \succ_a M(a)$  and  $p$  is either under-subscribed in  $M$  or there exists at least one agent  $a' \in M(p)$  such that  $a \succ_p a'$ . A matching  $M$  is *stable* if there is no blocking pair w.r.t.  $M$ .

It is well known that every HR instance admits a stable matching and all stable matchings are of the same size [11]. Size of a matching plays a very important role in real-world applications where leaving agents unmatched is undesirable and sometimes even unacceptable. In

applications like school choice [2] every child must find a school. In case of matching sailors to billets in the US Navy [26,23], every sailor must be assigned to some billet, apart from some additional constraints. Relaxing stability in order to enable larger size matchings has been investigated in literature [13,5]. In all such works, there is an inherent assumption that the quotas of programs are *rigid* and cannot be compromised.

However, there are real-world applications where quotas are determined by logistic considerations like resource availability, classroom size and these may be flexible. For instance, every semester elective allocation for under-graduate students at an educational institute happens via an automated procedure and once the preferences are available to the academic office, course instructors are consulted to adjust class capacities if appropriate. A recent work by Gajulapalli et al. [10] studies the school choice problem in a two round-setting – in the first round the quotas given as input are considered rigid. In the second round though, some schools may increase their quota as suggested by the mechanism in order to match *all* the students in a specific set in a *stability preserving* manner.

Motivated by applications where  $\mathcal{A}$ -perfectness (matching *every* agent) is mandated but is impossible to achieve in the presence of rigid quotas, we introduce and study the setting where costs control the number of agents matched to a program. We denote this as the Cost Controlled Quota (CCQ) setting. An instance  $H = (\mathcal{A} \cup \mathcal{P}, E)$  in the CCQ setting is similar to an HR instance except that programs do not have input quotas, instead a program  $p$  has a finite, non-negative integer cost  $c(p)$  denoting the cost of matching an agent to  $p$ . A matching  $M \subseteq E$  in the CCQ setting is an assignment of agents to programs where a program  $p$  can be matched to as many agents as possible in  $\mathcal{N}(p)$  and the number of agents assigned is controlled by costs. If a program is not assigned any agents in a matching, we call the program as being *closed*. Throughout, our goal is to compute an  $\mathcal{A}$ -perfect matching which is optimal with respect to preferences as well as costs. We remark that a very recent work by Santhini et al. [1] studies the cost-based setting for matchings with one-sided preferences. However, the problems considered in [1] are very different from ours.

We first define the notion of optimality in the CCQ setting w.r.t. the preferences and subsequently with respect to the costs. Stability is inherently defined using the input quotas. Envy-freeness [25], a relaxation of stability, is a natural substitute since it is defined *without* input quotas. In absence of input quotas, stability is equivalent to envy-freeness.

**Definition 2 (Envy-free matchings).** *Given a matching  $M$ , an agent  $a$  has a justified envy (here onwards called *envy*) towards a matched agent  $a'$ , where  $M(a') = p$  and  $(a, p) \in E$  if  $p \succ_a M(a)$  and  $a \succ_p a'$ . The pair  $(a, a')$  is an *envy-pair* w.r.t.  $M$ . A matching  $M$  is *envy-free* if there is no *envy-pair* w.r.t.  $M$ .*

Given a CCQ instance, our goal is to compute an  $\mathcal{A}$ -perfect envy-free matching subject to the following two optimization criteria with respect to costs.

1. **Minimize the total cost:** The total cost of a matching  $M$  is defined as  $\sum_{p \in \mathcal{P}} (|M(p)| \cdot c(p))$ . Our goal is to compute an  $\mathcal{A}$ -perfect envy-free matching that minimizes the total cost – we denote this as the MINSUM problem.
2. **Minimize the maximum cost:** The maximum cost spent at a program for a matching  $M$  is defined as  $\max_{p \in \mathcal{P}} \{|M(p)| \cdot c(p)\}$ . Our goal is to compute an  $\mathcal{A}$ -perfect envy-free matching that minimizes the maximum cost – we denote this as the MINMAX problem.

Consider the CCQ instance  $H$  in Fig. 1 with five agents and four programs. The preferences of the agents and the programs are as given in the figure. The number in the bracket beside a program denotes the cost of matching an agent to the respective program. The matching  $M$  shown in the figure is  $\mathcal{A}$ -perfect, as well as envy-free. The total cost of  $M$  is 10 and the max-cost of  $M$  is 6. It is easy to verify that in the instance  $H$ , the matching  $M$  is an optimal solution for both the MINSUM as well as the MINMAX problem; this need not be true in general. The MINSUM can be viewed as a global objective whereas the MINMAX is a local objective function.

|                                 |                                               |                                                                      |
|---------------------------------|-----------------------------------------------|----------------------------------------------------------------------|
| $a_1 : p_1 \succ p_0$           | (0) $p_0 : a_1 \succ a_2 \succ a_3 \succ a_4$ | $M = \{(a_1, p_1), (a_2, p_1), (a_3, p_1), (a_4, p_1), (a_5, p_2)\}$ |
| $a_2 : p_1 \succ p_0$           | (1) $p_1 : a_1 \succ a_2 \succ a_3 \succ a_4$ |                                                                      |
| $a_3 : p_1 \succ p_0$           | (6) $p_2 : a_4 \succ a_5$                     |                                                                      |
| $a_4 : p_1 \succ p_2 \succ p_0$ | (11) $p_3 : a_5$                              |                                                                      |
| $a_5 : p_2 \succ p_3$           |                                               |                                                                      |

**Fig. 1.** A CCQ instance

**Our Contributions:** This is the first work that investigates the cost-controlled quotas under two-sided preferences. We present an efficient algorithm for the MINMAX problem whereas in a sharp contrast, the MINSUM turns out to be NP-hard under severe restrictions. To address the hardness, we present a novel linear program for the MINSUM problem and give primal dual approximation algorithm for a special case. We state our results formally below. The results marked with  $(\star)$  have their proofs in the Appendix.

**Theorem 1.** *The MINMAX problem is solvable in  $O(m \log m)$  time where  $m = |E|$ .*

**Theorem 2  $(\star)$ .** *The following hardness results hold for the MINSUM problem.*

1. *The MINSUM problem is strongly NP-hard even when every agent has a preference list of length exactly  $f \geq 2$ , there is a master list ordering on agents and programs and the instance has two distinct costs.*
2. *The MINSUM problem cannot be approximated within a factor  $\frac{7}{6} - \epsilon, \epsilon > 0$  even when the instance has three distinct costs, unless  $P = NP$ .*

We say that the instance has a *master list* [17] on agents if there exists a fixed ordering of agents such that the preference lists of every program obeys that ordering. A similar definition holds for a master list on programs. In the CCQ setting since  $\mathcal{A}$ -perfectness is guaranteed, it is natural for agents to submit short preference lists. However, since there is a guarantee that every agent is matched, the central authority is likely to impose a minimum requirement on the length of the preference list [14]. We further note that it is easy to compute the optimal solution for the MINSUM problem under the two extreme scenarios – when the preference lists are unit length or when the preference lists are complete.

We complement our hardness results with the following approximation algorithms. Let  $\text{CCQ}_{c_1, c_2}$  denote the CCQ instance with two distinct costs  $c_1$  and  $c_2$  such that  $0 \leq c_1 < c_2$ . Let  $\ell_a$  (resp.  $\ell_p$ ) denote the length of the longest preference list of an agent (resp. a program).

**Theorem 3.** *The MINSUM problem admits an  $\ell_a$ -approximation algorithm on  $\text{CCQ}_{c_1, c_2}$  instances.*

We achieve our approximation guarantee via a technically involved linear programming (LP) based algorithm and show that our analysis is tight. We remark that our LP is for the general MINSUM problem but the approximation guarantee is for the restricted setting. In many real-world applications,  $\ell_a$  is typically small, in many cases a constant [16,15].

Finally, we present two simple approximation algorithms for MINSUM on arbitrary instances.

**Theorem 4.** *The MINSUM problem admits the following approximation algorithms.*

1. *a linear time  $\ell_p$ -approximation algorithm.*
2. *a  $|\mathcal{P}|$ -approximation algorithm.*

Our  $\ell_p$ -approximation algorithm is based on a natural lower-bound on the MINSUM problem. We also present a family of instances which shows that  $\ell_p$  is the best guarantee that can be achieved using the particular lower-bound. We establish that an optimal solution of the MINMAX problem also serves as a lower-bound for the MINSUM problem on the same instance and this gives us the  $|\mathcal{P}|$ -approximation algorithm.

**Relation to other Models.** There are alternate ways to formulate an optimization problem in the CCQ setting: (i) given a total budget  $\mathcal{B}$ , compute a largest envy-free matching with cost at most  $\mathcal{B}$ . (ii) given an HR instance and a cost for every program, *augment* the input quotas to compute an  $\mathcal{A}$ -perfect envy-free matching with *optimal* cost. The NP-hardness for both these problems can be proved by easily modifying the NP-hardness reduction for MINSUM. As mentioned earlier, Gajulapalli et al. [10] consider the school choice problem in a two-round setting. In the second round, their goal is to match *all* agents in a particular set derived from the matching in the first round and they need to match them in an envy-free manner (called stability preserving in their work). The CCQ setting generalizes the matching problem in round-2 as follows: If  $c(p)$  denotes the cost of matching an agent to a program in round-2 then MINSUM computes an  $\mathcal{A}$ -perfect envy-free matching where the total cost of matching agents in round-2 is minimized. Let  $d(p)$  denote the deviation of program  $p$ , that is the additional number of agents matched to  $p$  in round-2 beyond its input quota  $q(p)$ . Then by setting  $c(p) = 1$  for every program, the MINMAX problem computes an  $\mathcal{A}$ -perfect envy-free matching in round-2 such that maximum deviation of a program is minimized. We remark that in [10] the authors state that a variant of MINSUM problem (Problem 33, Section 7) is NP-hard. However, they do not investigate the problem in detail.

**Other Related Work.** For a review of results related to stable matchings in the HR setting, we refer the reader to [11,12]. Envy-freeness (also called as fairness) is a well-studied notion of optimality. Structural properties of envy-free matchings in the HR setting are investigated in [25]. In the HRLQ setting (HR setting wherein hospitals have *lower quotas*) envy-free matchings are studied in [27,19,9].

We review the work where quotas are flexible or replaced by other constraints. *Flexible quotas* in the college admission setting are studied in [22]. In their setting, no costs are involved but colleges may have ties in the preference lists and flexible quotas are used for tie-breaking at the last matched rank. In the student-project allocation setting, the problem of minimizing the maximum and total deviation from the initial target is studied in [7]. A setting where courses make monetary transfers to students and have budget constraints is studied in [18] in which they propose a new notion of optimality, namely *approximate stability*. Funding constraints are studied in [3] in the context of allocating student interns to the projects funded by supervisors.

As mentioned earlier Santhini et al. [1] consider a cost-based quota setting which is same as ours. However, their problems are in the one-sided preference list setting and they show an efficient algorithm to compute a min-cost matching of a desired *signature*. The signature allows to encode requirements about the number of agents matched to a particular rank. This result is in contrast to the hardness results we show for similar optimization problems in this paper under the two-sided preference setting.

*Organization of the paper:* In Section 2 we present a linear programming formulation for MINSUM problem. In Section 3 we give an LP based approximation algorithm for MINSUM on  $\text{CCQ}_{c1,c2}$  instances. In Section 4, we present algorithmic results for MINMAX and MINSUM for general instances.

## 2 A Linear programming formulation for MINSUM

In this section we present a linear program (LP) for the general MINSUM problem and use the LP for obtaining an  $\ell_a$ -approximation algorithm for  $\text{CCQ}_{c1,c2}$ .

## 2.1 Linear Program and its dual

**Primal: minimize**

$$\sum_{p \in \mathcal{P}} c(p) \cdot \sum_{(a,p) \in E} x_{a,p} \quad (1)$$

**subject to**

$$\sum_{\substack{p': \\ p'=p \text{ or } \\ p' \succ_{a'} p}} x_{a',p'} \geq x_{a,p}, \quad \forall (a',p) \in E, a \prec_p a' \quad (2)$$

$$\sum_{(a,p) \in E} x_{a,p} = 1, \quad \forall a \in \mathcal{A} \quad (3)$$

$$x_{a,p} \geq 0, \quad \forall (a,p) \in E \quad (4)$$

**Dual: maximize**

$$\sum_{a \in \mathcal{A}} y_a \quad (5)$$

**subject to**

$$y_a + \sum_{\substack{p': \\ p'=p \text{ or } \\ p' \prec_{a'} p}} \sum_{\substack{a': \\ a' \prec_{p'} a}} z_{a',p',a} - \sum_{\substack{a': \\ a \prec_p a'}} z_{a',p,a} \quad (6)$$

$$\leq c(p), \quad \forall (a,p) \in E$$

$$z_{a',p,a} \geq 0, \quad \forall (a',p) \in E, a \prec_p a' \quad (7)$$

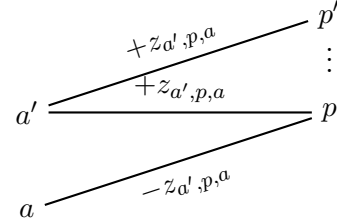
**Fig. 2.** Linear Program and its dual for the MINSUM problem

Fig. 2 shows the LP for the MINSUM problem. Let  $H = (\mathcal{A} \cup \mathcal{P}, E)$  be a CCQ instance. Let  $x_{a,p}$  be a primal variable for the edge  $(a,p) \in E$ :  $x_{a,p}$  is 1 if  $a$  is matched to  $p$ , 0 otherwise. The objective of the primal LP (Eq. 1) is to minimize the total cost of all matched edges. Eq. 2 encodes the envy-freeness constraint: if agent  $a$  is matched to  $p$  then every agent  $a' \succ_p a$  must be matched to either  $p$  or a higher-preferred program than  $p$ , otherwise  $a'$  envies  $a$ . In the primal LP, the envy-freeness constraint is present for a triplet  $(a',p,a)$  where  $a' \succ_p a$ . We call such a triplet a *valid* triplet. Eq. 3 encodes  $\mathcal{A}$ -perfectness constraint.

In the dual LP, we have two kinds of variables, the  $y$  variables which correspond to every agent and the  $z$  variables that correspond to every valid triplet in the primal program. The dual constraint (Eq. 6) is for every edge  $(a,p)$  in  $E$ . The  $y_a$  variable corresponding to an agent  $a$  appears in the dual constraint corresponding to every edge incident on  $a$ . The value  $y_a$  can be interpreted as the cost paid by agent  $a$  for matching  $a$  to one of the programs in  $\mathcal{N}(a)$ . For an edge  $(a,p)$  and an agent  $a' \succ_p a$ , the dual variable  $z_{a',p,a}$  appears in negative form in exactly one constraint and it is for the edge  $(a,p)$ . The same dual variable  $z_{a',p,a}$  appears in positive form in the constraint for every edge  $(a',p')$  such that  $p' = p$  or  $p' \succ_{a'} p$ . (refer Fig. 3). The value of  $z_{a',p,a}$  can be interpreted as the cost paid by agent  $a$  in matching  $a'$  to a program  $p'$  such that  $p' = p$  or  $p' \succ_{a'} p$  to resolve potential envy-pair  $(a',a)$ . Following are some useful facts about the linear program.

**Fact 1.** Let  $a$  be a fixed agent. If  $y_a$  is incremented by a positive value  $\Delta$  then it increments the left-hand side (lhs) of the dual constraint of *every* edge  $(a,p)$  by  $\Delta$  and it does not affect the dual constraint of any edge incident on agent  $a' \neq a$ .

**Fact 2.** Let  $(a',p,a)$  be a fixed valid triplet. If  $z_{a',p,a}$  is incremented by a positive value  $\Delta$  then it increments the lhs of the dual constraint of *every* edge  $(a',p')$  by  $\Delta$  such that  $p' = p$  or  $p' \succ_{a'} p$ , reduces the lhs of the dual constraint of *exactly one* edge  $(a,p)$  by  $\Delta$  and does not affect the dual constraint of any edge incident on agent  $a'' \notin \{a, a'\}$ .



**Fig. 3.** Let  $(a',p,a)$  be a valid triplet and  $p' \succ_{a'} p$ . The edges shown in the figure are those whose dual constraint contains the variable  $z_{a',p,a}$  in either positive or negative form.

The following notation is used in illustrating the challenges and in our  $\ell_a$ -approximation algorithm in the next section. For a given dual setting and an edge, if Eq. 6 is satisfied with equality then we call such an edge a *tight edge*, otherwise it is a *slack edge*. For an edge  $(a, p)$ ,  $\text{slack}(a, p)$  denotes its slack. When referring to a  $z$  variable, when a specific agent or program occurring in it does not matter, we use  $\times$  in its place.

**Definition 3 (Threshold agent).** *Let  $M$  be a matching in the instance  $H$ . For every program  $p$ ,  $\text{thresh}(p)$  is the most-preferred agent  $a$ , if it exists, such that  $p \succ_a M(a)$ , otherwise  $\text{thresh}(p)$  is  $\perp$ .*

The definition of threshold agent is similar to the threshold resident defined in [20] and a barrier (vertex) defined in [10]. We remark that the threshold agent depends on the matching  $M$ , hence when  $M$  gets modified, the threshold agents for programs may change.

**Definition 4 (Matchable edge).** *For an envy-free matching  $M$ , and an agent  $a$  (matched or unmatched), we say that an edge  $(a, p) \notin M$  is matchable if the dual constraint on  $(a, p)$  is tight and  $a = \text{thresh}(p)$ , otherwise the edge is non-matchable.*

It is straightforward to verify that for an envy-free matching  $M$ , if we match agent  $a$  along a matchable edge then the resultant matching remains envy-free.

## 2.2 Challenges

A standard primal-dual approach for the MINSUM problem would be to begin with a dual feasible solution. The algorithm then repeatedly updates the dual till we obtain a primal feasible solution using the tight edges w.r.t. to the dual setting. We illustrate the challenges in using such an approach for the general MINSUM problem. Consider the MINSUM instance in Fig. 1. Assume that we begin with an initial dual setting where all dual variables are set to 0. The matching  $M = \{(a_1, p_0), (a_2, p_0), (a_3, p_0), (a_4, p_0)\}$  obtained on the tight edges is envy-free but does not match agent  $a_5$  and hence is not primal feasible. Since no edge incident on  $a_5$  is tight (slack on  $(a_5, p_2)$  and  $(a_5, p_3)$  is 6 and 11 respectively) we can set  $y_5$  to 6 while maintaining dual feasibility. We note that while this update makes the edge  $(a_5, p_2)$  tight, adding the edge to the matching  $M$  introduces an envy pair – namely  $a_4$  envying  $a_5$ . We note that this is the our first difficulty, that is, while there are tight edges incident on an unmatched agent, none of them may be matchable.

The second difficulty stems from the following: in order to address the potential envy pair  $(a_4, a_5)$ , we must *promote* agent  $a_4$ . With the current dual setting,  $y_4$  cannot be increased hence a natural way is to update a  $z$  variable. This can indeed be achieved by setting  $z_{a_4, p_2, a_5} = 1$ , thus making  $(a_4, p_1)$  tight. However, as encountered earlier, this edge is not matchable, since matching  $a_4$  to  $p_1$  introduces several other envy pairs. Note that this chain of potential envy resolutions is triggered by the unmatched agent  $a_5$ . Since, this chain can be arbitrarily long it is not immediate if the update in  $z$  variable can be charged to an update in some  $y$  variable, thereby achieving a reasonable approximation ratio.

## 3 An $\ell_a$ -approximation algorithm for MINSUM on $\text{CCQ}_{c_1, c_2}$

In this section we show that when the  $\text{CCQ}$  instance has only two distinct costs  $c_1$  and  $c_2$ , we are able to circumvent the challenges and obtain an  $\ell_a$ -approximation algorithm for the MINSUM problem. We recall from Theorem 2 that even in this restricted setting, the problem remains NP-hard.

**High-level idea of the algorithm.** Our LP based algorithm begins with an initial feasible dual setting and an envy-free matching  $M$  which need not be  $\mathcal{A}$ -perfect. As long as  $M$  is not  $\mathcal{A}$ -perfect, we pick an unmatched agent  $a$  and increase the dual variable  $y_a$ . We show that for an unmatched agent such an increase is possible and *all* edges incident on  $a$  become tight due to the

update. However, none of the edges incident on  $a$  may be matchable (since for every  $p \in \mathcal{N}(a)$ ,  $\text{thresh}(p) \neq a$ ). Under the restricted setting we ensure that after a bounded number of updates to the  $z$  variables, at least one edge incident on  $a$  is matchable. Throughout we maintain the following invariants with respect to the matching  $M$ .

- $M$  is envy-free, not necessarily  $\mathcal{A}$ -perfect and every matched edge is tight.
- For an agent  $a$  (matched or unmatched), for every  $p \succ_a M(a)$ , either (i)  $(a, p)$  is tight and  $\text{thresh}(p) \neq a$  or (ii)  $\text{slack}(a, p) = c_2 - c_1$ .

We remark that when the matching is modified, thresholds may change, due to which a tight, non-matchable edge may become matchable. As long as there exists such an edge, we match it. This is achieved by the *free-promotions* routine which can be implemented in  $O(m^2)$  time. We defer the formal description and analysis of this routine to the end of the section.

**Description of the algorithm.** Algorithm 1 gives the pseudo-code. In Appendix A, we give an illustrative example which depicts the key steps of the algorithm on a  $\text{CCQ}_{c_1, c_2}$  instance. We begin with an empty matching  $M$  and by setting all  $y$  variables to  $c_1$  and all  $z$  variables to 0 (line 1). Following this, for every agent  $a$  with a cost  $c_1$  program in  $\mathcal{N}(a)$  we match the agent to its most-preferred program with cost  $c_1$  (**for** loop at line 2). Next, we compute the threshold agent for every program w.r.t.  $M$ . As long as  $M$  is not  $\mathcal{A}$ -perfect, we pick an arbitrary unmatched agent  $a$  and update the dual variables as follows.

---

**Algorithm 1** Algorithm to compute an  $\ell_a$ -approximation of MINSUM on  $\text{CCQ}_{c_1, c_2}$

---

```

1: let  $M = \emptyset$ , all  $y$  variables are set to  $c_1$  and all  $z$  variables are set to 0
2: for every agent  $a \in \mathcal{A}$  s.t.  $\exists p \in \mathcal{N}(a)$  such that  $c(p) = c_1$  do
3:   let  $p$  be the most-preferred program in  $\mathcal{N}(a)$  s.t.  $c(p) = c_1$  and let  $M = M \cup \{(a, p)\}$ 
4: compute  $\text{thresh}(p)$  for every program  $p \in \mathcal{P}$ 
5: while  $M$  is not  $\mathcal{A}$ -perfect do
6:   let  $a$  be an unmatched agent
7:   while  $a$  is unmatched do
8:     set  $y_a = y_a + c_2 - c_1$ 
9:     if there exists a matchable edge incident on  $a$  then
10:       $M = M \cup \{(a, p) \mid (a, p) \text{ is the most-preferred matchable edge for } a\}$ 
11:      perform free-promotions routine
12:    else
13:       $\mathcal{P}(a) = \{p \in \mathcal{N}(a) \mid p \succ_a M(a), (a, p) \text{ is tight and } \text{thresh}(p) \neq a\}$ 
14:      while  $\mathcal{P}(a) \neq \emptyset$  do
15:        let  $a'$  be the threshold agent of some program in  $\mathcal{P}(a)$ 
16:        let  $\mathcal{P}(a, a')$  denote the set of programs in  $\mathcal{P}(a)$  whose threshold agent is  $a'$ 
17:        let  $p$  be the least-preferred program for  $a'$  in  $\mathcal{P}(a, a')$ 
18:        set  $z_{a', p, a} = c_2 - c_1$ 
19:        let  $(a', p')$  be the most-preferred matchable edge incident on  $a'$ . Unmatch  $a'$  if matched and let
            $M = M \cup \{(a', p')\}$ 
20:        execute free-promotions routine, re-compute thresholds and the set  $\mathcal{P}(a)$ 
21: return  $M$ 

```

---

1. For the agent  $a$ , we increase  $y_a$  by  $c_2 - c_1$ . We ensure that the dual setting is feasible and all edges incident on  $a$  become tight for the dual constraint in Eq. 6. Although this step makes all edges incident on  $a$  tight, they may not be necessarily matchable. Recall that a tight edge  $(a, p)$  is matchable if  $\text{thresh}(p) = a$ .
2. If there is a program  $p$  such that  $(a, p)$  is matchable, then  $a$  is immediately matched to the most-preferred such program  $p$  (line 10) and we are done with matching agent  $a$ . Since the matching is modified, we execute free-promotions routine.
3. In case there is no such program for which  $a$  is the threshold agent, we update carefully selected  $z$  variables in order to either promote the threshold agent (if matched) or match the threshold agent.

- (a) We compute the set  $\mathcal{P}(a)$  of programs  $p \in \mathcal{N}(a)$  such that the dual constraint on edge  $(a, p)$  is tight and  $\text{thresh}(p) \neq a$  and  $p \succ_a M(a)$  (line 13). In other words,  $\mathcal{P}(a)$  is the set of programs in the neighbourhood of  $a$  such that  $p$  is higher-preferred over  $M(a)$  and edge  $(a, p)$  is tight but not matchable.
- (b) By the definition of  $\mathcal{P}(a)$ , for every  $p_j \in \mathcal{P}(a)$ , there exists  $\text{thresh}(p_j) = a' \neq a$ . We pick an arbitrary agent  $a'$  that is a threshold of some program in  $\mathcal{P}(a)$  (line 15). Note that the agent  $a'$  can be the threshold agent of more than one program in  $\mathcal{P}(a)$ , and we let  $\mathcal{P}(a, a')$  denote the set of programs in  $\mathcal{P}(a)$  for whom  $a'$  is the threshold. Let  $p$  be the least-preferred program for  $a'$  in  $\mathcal{P}(a, a')$  (line 17).
- (c) Our goal is to match  $a'$  to a program  $p'$  such that  $p' = p$  or  $p' \succ_{a'} p$ . By the choice of  $a, a'$  and  $p$  and from the primal LP,  $(a', p, a)$  is a valid triplet and therefore there exists a dual variable  $z_{a', p, a}$  (refer Fig. 3). We set  $z_{a', p, a}$  to  $c_2 - c_1$  (line 18). We ensure that this update maintains dual feasibility.
- (d) Recall that the variable  $z_{a', p, a}$  appears in the positive form in the dual constraint of every edge  $(a', p')$  such that  $p' = p$  or  $p' \succ_{a'} p$ . We ensure that this update results in making all edges  $(a', p')$  tight and at least one of these becomes matchable. We match  $a'$  along the most-preferred matchable edge (line 19). Recall that  $z_{a', p, a}$  variable appears in negative form in the dual constraint of edge  $(a, p)$ , hence edge  $(a, p)$  becomes slack after this update.
- (e) Since  $M$  is modified, we execute free-promotions routine. If a tight edge incident on  $a$  becomes matchable, then  $a$  is matched inside the free-promotions routine.
- (f) We remark that the set  $\mathcal{P}(a)$  computed in line 13 is dependent on the matching  $M$ , specifically  $M(a)$  and the threshold agents w.r.t.  $M$ . In order to maintain a specific slack value on the edges that is useful in maintaining dual feasibility and ensuring progress, we re-compute the set  $\mathcal{P}(a)$  (line 20) and re-enter the loop in line 14 if  $\mathcal{P}(a) \neq \emptyset$ .

**Observations.** We observe the following properties of the algorithm.

**(P1)** At line 4, no agent is assigned to any program with cost  $c_2$  and for every agent  $a$  (matched or unmatched), every program  $p \succ_a M(a)$  has cost  $c_2$ .

**(P2)** A matched agent never gets demoted. This is clear by observing that whenever a matched agent  $a$  changes its partner from  $M(a)$  to program  $p$ , we have  $\text{thresh}(p) = a$ , hence by the definition of the threshold agent,  $p \succ_a M(a)$ .

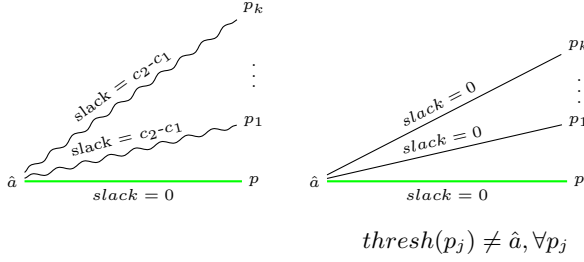
### 3.1 Proof of correctness

We first prove that matching  $M$  is envy-free.

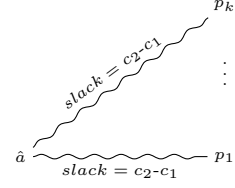
**Lemma 1.** *Matching  $M$  is envy-free throughout the execution of the algorithm.*

*Proof.* Matching  $M$  is trivially envy-free after line 1. Any two agents  $a$  and  $a'$  that are matched in line 3 are matched to a program with cost  $c_1$  and by the choice made in line 3, it is clear that they do not form an envy-pair. By **(P1)**, every unmatched agent  $a$  has only cost  $c_2$  programs in  $\mathcal{N}(a)$  thus, no unmatched agent envies an agent matched in line 3. Thus,  $M$  is envy-free before entering the loop at line 5.

Suppose  $M$  is envy-free before a modification in  $M$  inside the loop. We show that it remains envy-free after the modification. Matching  $M$  is modified either at line 10 or line 19 or inside the free-promotions routine. In all these places, only a matchable edge  $(a_i, p_j)$  is matched. Therefore no agent  $a' \neq a_i$  envies  $a_i$  after this modification. Before this modification  $a_i$  did not envy  $a' \neq a_i$  and by **(P2)**  $a_i$  (if matched) is not demoted, therefore  $a_i$  does not envy  $a' \neq a_i$  after the modification. Thus,  $M$  remains envy-free.  $\square$



**Fig. 4.** Matched agent  $\hat{a}$ , for every  $p_j \succ_{\hat{a}} p = M(\hat{a})$ : one of these two conditions hold



**Fig. 5.** Unmatched agent  $\hat{a}$ , for every  $p_j \in \mathcal{N}(\hat{a})$

Recall that if edge  $(\hat{a}, \hat{p})$  is non-matchable then either  $(\hat{a}, \hat{p})$  is slack or  $\text{thresh}(\hat{p}) \neq \hat{a}$ . In Lemma 2, we establish important properties about *every* non-matchable edge incident on an agent  $\hat{a}$  that is higher-preferred over  $M(\hat{a})$ . These properties (Fig. 4 and Fig. 5) are crucial in showing dual feasibility, progress and approximation guarantee.

**Lemma 2** ( $\star$ ). *Before every iteration of the loop starting at line 7, the following holds.*

- (i) *For a matched agent  $\hat{a}$ , either for every program  $p_j \succ_{\hat{a}} M(\hat{a})$ ,  $\text{slack}(\hat{a}, p_j) = c_2 - c_1$  or for every program  $p_j \succ_{\hat{a}} M(\hat{a})$ ,  $\text{slack}(\hat{a}, p_j) = 0$  and  $\text{thresh}(p_j) \neq \hat{a}$ .*
- (ii) *For an unmatched agent  $\hat{a}$ , for every program  $p_j \in \mathcal{N}(\hat{a})$ ,  $\text{slack}(\hat{a}, p_j) = c_2 - c_1$ .*

We observe an important property of the agent  $a'$  selected in line 15. Let  $a_j$  be an agent such that for every  $p_j \succ_{a_j} M(a_j)$ ,  $\text{slack}(p_j) = c_2 - c_1$  before an iteration of the loop at line 14. Suppose during this iteration,  $a' = a_j$  is selected in line 15 and let  $p = p_k$  is selected in line 17. Then by **Fact 2**, for every  $p_j$  such that  $p_j = p_k$  or  $p_j \succ_{a_j} p_k$ , the dual update in line 18 results in making all  $(a_j, p_j)$  edges tight. Also, since  $\text{thresh}(p_k) = a_j$ , at least one of these newly tight edges (specifically,  $(a_j, p_k)$ ) becomes matchable. Therefore,  $M(a_j)$  is modified. Agent  $a_j$  may further get promoted by the free-promotions routine and  $a_j$  remains matched such that for every  $p_j \succ_{a_j} M(a_j)$ ,  $\text{slack}(a_j, p_j) = 0$  and  $\text{thresh}(p_j) \neq a_j$ . Thus during the rest of the execution of this loop,  $a' \neq a_j$  at line 15. This observation implies the following corollary which is used to prove Lemma 2 and dual feasibility (Lemma 3).

**Corollary 1.** *Let  $a_j$  be an agent such that for every  $p_j \succ_{a_j} M(a_j)$ ,  $\text{slack}(p_j) = c_2 - c_1$  before an iteration of the loop at line 14. Then,  $a' = a_j$  is selected at most once inside the loop at line 14. Thus, exactly one variable of the form  $z_{a_j, \times, \times}$  is updated during the course of the algorithm. When the loop terminates,  $a_j$  remains matched such that for every  $p_j \succ_{a_j} M(a_j)$ ,  $\text{slack}(a_j, p_j) = 0$  and  $\text{thresh}(p_j) \neq a_j$ .*

**Lemma 3.** *The dual setting is feasible throughout the algorithm.*

*Proof.* It is clear that the dual setting is feasible before entering the loop after line 4. We show that if the dual setting is feasible before an arbitrary dual update (either line 8 or line 18) then it remains feasible after the update.

- **Update at line 8:** Since  $a$  is unmatched, by Lemma 2, the slack on every edge incident on  $a$  is  $c_2 - c_1$ . By **Fact 1**, this update increases the lhs of every edge incident on  $a$  by  $c_2 - c_1$ . Therefore the dual setting is feasible.
- **Update at line 18:** Let  $a' = a_j$  is selected at line 15 and  $p \succ_{a_j} M(a_j)$  be selected at line 17. If this is the first iteration of the loop at line 14 during which  $a' = a_j$  is selected then by the choice of  $a'$  and Lemma 2, for every  $p_j \succ_{a_j} M(a_j)$ , the slack is  $c_2 - c_1$ . By **Fact 2**, the update in line 18 increases the lhs of every edge  $(a_j, p_j)$  such that  $p_j = p$  or  $p_j \succ_{a_j} p$  by  $c_2 - c_1$ , maintaining the dual feasibility. If this is not the first iteration of the loop at line 14 then it contradicts the choice of  $a' = a_j$  since Corollary 1 implies that  $a' = a_j$  is selected at most once inside the loop at line 14.

This completes the proof of the lemma.  $\square$

We prove that the algorithm terminates by computing an  $\mathcal{A}$ -perfect matching  $M$  (Lemma 4) and that  $M$  is an  $\ell_a$ -approximation (Lemma 5).

**Lemma 4.** *Algorithm 1 computes an  $\mathcal{A}$ -perfect matching.*

*Proof.* We first show that in every iteration of the loop in line 7, either an unmatched agent is matched or at least one agent is promoted: By Lemma 2 and **Fact 1**, the dual update in line 8 all edges incident on  $a$  become tight. Either  $a$  gets matched in line 10 or the loop in line 14 executes at least once. Since  $\mathcal{P}(a) \neq \emptyset$  every time the loop at line 14 is entered, an agent  $a'$  is selected in line 15. By the choice of  $a'$ , Lemma 2, **Fact 2** and the choice of  $p$  in line 17, the dual update in line 18 ensures that at least one edge  $(a', p_j)$ , for  $p_j \succ_{a'} M(a')$  becomes matchable and  $a'$  gets matched along that edge. By **(P2)**, this modification does not demote  $a'$  (if  $a'$  was already matched). Therefore, either an unmatched agent is matched or at least one agent is promoted during an iteration.

Thus after  $O(m)$  iterations of the loop in line 7, a fixed unmatched agent  $a$  gets matched and the loop in line 7 terminates. As mentioned earlier, the free-promotions routine takes  $O(m^2)$  time. Thus, the loop in line 7 terminates in  $O(m^3)$  time for a fixed unmatched agent  $a$  and the loop in 5 terminates in  $O(m^3|\mathcal{A}|)$  time. By the termination condition of the loop,  $M$  is an  $\mathcal{A}$ -perfect matching.  $\square$

**Lemma 5**  $(\star)$ . *Matching  $M$  computed by Algorithm 1 is an  $\ell_a$ -approximation of MINSUM.*

*Proof (sketch).* The cost of matching  $M$  is given by the primal objective function (Eq. 1). We show that every matched edge is tight. This enables us to write the cost of  $M$  in terms of the lhs of the dual constraint (Eq. 6) of the matched edges. The cost of  $M$  contains two components – the contribution of  $y$  variables and the contribution of  $z$  variables. We *charge* the update in every  $z$  variable to an appropriately chosen  $y$  variable.

In our analysis, we use the following observation – recall that a variable  $z_{a',p,a}$  exists for a valid triplet  $(a', p, a)$ . If there exists  $p'$  such that  $p' = p$  or  $p' \succ_{a'} p$  and  $(a', p') \in M$  and  $(a, p) \in M$ , then the  $z_{a',p,a}$  variable cancels out in the total cost of the matching. We carefully analyse these cases and upper-bound the total contribution of  $z$  variables that are not cancelled. We show that the contribution of such  $z$  variables is at most  $(\ell_a - 1) \sum_{a \in \mathcal{A}} y_a$ . Combining this with the contribution of  $y$  variables we show that  $M$  is an  $\ell_a$ -approximation.  $\square$

We conclude this section by describing the free-promotions routine: The free-promotions routine checks if there exists a matchable edge  $(a, p)$ . If there is no such edge, the routine terminates. Otherwise, it matches  $(a, p)$ , re-computes the threshold agents and repeats the search. Checking for a matchable edge and computing threshold agents takes  $O(m)$  time. Thus, the free-promotions routine runs in  $O(m^2)$  time.

This establishes Theorem 3. In Appendix B.1 we show that our analysis is tight. The restriction of two distinct costs is crucially used in the analysis of our approximation algorithm for MINSUM on  $\text{CCQ}_{c1,c2}$  instances. An interesting open question is to use the LP for general instances.

## 4 MINSUM on arbitrary instances and MINMAX

In this section, we present our algorithmic results for MINSUM problems on arbitrary CCQ instances and for the MINMAX problem.

#### 4.1 $\ell_p$ -approximation for MINSUM

Let  $H$  be a CCQ instance and let  $p_a^*$  denote the minimum cost program in the preference list of agent  $a$ . If there is more than one program with the same minimum cost, we let  $p_a^*$  be the most-preferred such program. Our algorithm (Algorithm 2) starts by matching every agent  $a$  to  $p_a^*$ . Note that such a matching is  $\mathcal{A}$ -perfect and min-cost but not necessarily envy-free. Now the algorithm considers programs in an arbitrary order. For program  $p$ , we consider agents in the reverse preference list ordering of  $p$ . If there exists agent  $a \notin M(p)$  such that  $p \succ_a M(a)$  and there exists  $a' \in M(p)$  such that  $a' \prec_p a$ , then  $(a, a')$  form an envy-pair. We resolve this by promoting  $a$  from  $M(a)$  to  $p$ . The algorithm stops after considering every program.

---

**Algorithm 2** An  $\ell_p$ -approximation algorithm for MINSUM
 

---

```

1: let  $M = \{(a, p) \mid a \in \mathcal{A} \text{ and } p = p_a^*\}$ 
2: for every program  $p$  do
3:   for  $a$  in reverse preference list ordering of  $p$  do
4:     if there exists  $a' \in M(p)$  such that  $a \succ_p a'$  and  $p \succ_a M(a)$  then
5:        $M = M \setminus \{(a, M(a))\} \cup \{(a, p)\}$ 
6: return  $M$ 
    
```

---

Note that in Algorithm 2 an agent may get promoted in the loop (line 5) of the algorithm but never gets demoted. Further, program  $p$  is assigned agents in the **for** loop (line 2) only when at least one agent is matched to  $p$  in line 1. Therefore, if program  $p$  is assigned at least one agent in the final output matching, then  $p = p_a^*$  for some agent  $a \in \mathcal{A}$ .

**Analysis:** It is clear that the matching computed by Algorithm 2 is  $\mathcal{A}$ -perfect. Now we show that output of the algorithm is an envy-free matching and an  $\ell_p$ -approximation to an optimal solution of the MINSUM problem.

**Lemma 6.** *The matching  $M$  output by Algorithm 2 is envy-free.*

*Proof.* We show that no agent participates in an envy-pair w.r.t.  $M$ . Assume for contradiction, that  $(a, a')$  form an envy-pair w.r.t.  $M$ . Suppose  $p = M(a')$ . Then  $p \succ_a M(a)$  and  $a' \prec_p a$ . Consider the iteration of the **for** loop in line 2 when  $p$  was considered. Agent  $a'$  was either already matched to  $p$  (before the iteration began) or is assigned to  $p$  in this iteration. Note that  $a \succ_p a'$  and the agents are considered in reverse order of  $p$ 's preferences. Thus, in either case when  $a$  was considered in line 3,  $a' \in M(p)$ . If  $M(a) = p$  or  $M(a) \succ_a p$  at this line then  $M(a) = p$  or  $M(a) \succ_a p$  at the end of the algorithm, since an agent never gets demoted. Thus, we must have that  $M(a) \prec_a p$  at this line. This implies that the algorithm matched  $a$  to  $p$ . Since  $a$  can only get promoted during the subsequent iterations,  $M(a) = p$  or  $M(a) \succ_a p$  at the end of the algorithm. This contradicts the claimed envy.  $\square$

Next we show the approximation guarantee using a natural lower-bound on the optimal cost of MINSUM. Let  $\text{OPT}$  denote an optimal solution and  $c(\text{OPT})$  denote the cost of  $\text{OPT}$ . Since  $\text{OPT}$  is  $\mathcal{A}$ -perfect, it is easy to observe that  $c(\text{OPT}) \geq \sum_{a \in \mathcal{A}} c(p_a^*)$ . We denote this lower-bound as  $lb_1$ . Using  $lb_1$  we show that the output matching  $M$  is an  $\ell_p$ -approximation.

**Lemma 7.** *The matching  $M$  output by Algorithm 2 is an  $\ell_p$ -approximation.*

*Proof.* Let  $c(M)$  denote the cost of matching  $M$ . In the matching  $M$ , some agents are matched to their least-cost program (call them  $\mathcal{A}_1$ ), whereas some agents get promoted (call them  $\mathcal{A}_2$ ). Recall that if a program  $p$  is assigned agents in  $M$  then  $p = p_a^*$  for some agent  $a$ . Thus for an agent  $a \in \mathcal{A}_2$ , we charge the cost of some other least-cost program  $p_{a'}^*$  such that  $a' \neq a$ . A program can be charged at most  $\ell_p - 1$  times for the agents in  $\mathcal{A}_2$ , thus

$$c(M) = \sum_{a \in \mathcal{A}_1} c(p_a^*) + \sum_{a \in \mathcal{A}_2} c(M(a)) \leq \sum_{a \in \mathcal{A}} c(p_a^*) + \sum_{a \in \mathcal{A}_2} (\ell_p - 1) \cdot c(p_a^*) \leq \ell_p \cdot c(\text{OPT})$$

This completes the proof of the lemma.  $\square$

This establishes Theorem 4.1. Although  $lb_1$  is a natural lower-bound, in Appendix C, we show that the best approximation guarantee using  $lb_1$  is  $\ell_p$ , thereby showing that the analysis of our algorithm is tight.

## 4.2 MINMAX problem and a $|\mathcal{P}|$ -approximation for MINSUM

Let  $H$  be a CCQ instance. Our algorithm for the MINMAX problem is based on the following observations: Let  $M^*$  be an optimal solution for the MINMAX problem on  $H$ .

- Let  $t^* = \max_{p \in \mathcal{P}} \{c(p) \cdot |M^*(p)|\}$ . Consider an HR instance  $G_{t^*}$  where the preference lists are borrowed from  $H$  and for each  $p \in \mathcal{P}$ ,  $q(p) = \left\lfloor \frac{t^*}{c(p)} \right\rfloor$ . Note that  $M^*$  is an  $\mathcal{A}$ -perfect envy-free matching in  $H$  such that  $|M^*(p)| \leq q(p)$  for every  $p \in \mathcal{P}$ . Thus the instance  $G_{t^*}$  admits an  $\mathcal{A}$ -perfect stable matching.
- For any  $t < t^*$ , consider an HR instance  $G_t$  with  $q(p) = \left\lfloor \frac{t}{c(p)} \right\rfloor$ . The instance  $G_t$  does not admit an  $\mathcal{A}$ -perfect stable matching. Otherwise this contradicts the optimality of  $M^*$ .
- The optimal value  $t^*$  lies in the range  $\min_{p \in \mathcal{P}} \{c(p)\}$  to  $\max_{p \in \mathcal{P}} \{(\text{len}(p) \cdot c(p))\}$ , where  $\text{len}(p)$  denotes the length of the preference list of program  $p$ .
- For any  $t' > t^*$ ,  $G_{t'}$  admits an  $\mathcal{A}$ -perfect envy-free matching.

Our algorithm begins by constructing a sorted array  $\hat{c}_p$  for each program  $p \in \mathcal{P}$  such that for  $1 \leq i \leq \text{len}(p)$ , we have  $\hat{c}_p[i] = i \cdot c(p)$ . There are  $|\mathcal{P}|$  many such sorted arrays and the total number of elements in these arrays is  $\sum_{p \in \mathcal{P}} \text{len}(p) = |E| = m$ . We merge these arrays to construct a sorted array  $\hat{c}$  of distinct costs. Then we perform a binary search for the optimal value of  $t^*$  in the sorted array  $\hat{c}$ : for a particular value  $t = \hat{c}[k]$  we construct the HR instance  $G_t$  by setting appropriate quotas. If the stable matching in  $G_t$  is not  $\mathcal{A}$ -perfect, then we search in the *upper-range*. Otherwise, we check if  $G_{t'}$  admits an  $\mathcal{A}$ -perfect stable matching for  $t' = \hat{c}[k - 1]$ . If not, we return  $t$  otherwise, we search for the optimal in the *lower-range*.

**Running time.** The sorted array  $\hat{c}$  is constructed by merging  $|\mathcal{P}|$  sorted arrays containing  $m$  total entries. Thus, this step takes  $O(m \log |\mathcal{P}|)$  time. The algorithm requires  $O(\log m)$  iterations because there are at most  $m$  distinct costs in  $\hat{c}$ . Each iteration computes at most two stable matchings using the linear time algorithm [11]. Thus the overall running time is  $O(m \log m)$ . This establishes Theorem 1.

We now prove that the optimal matching  $M^*$  for the MINMAX problem is a  $|\mathcal{P}|$ -approximation for the MINSUM problem.

**Lemma 8.** *The optimal solution for the MINMAX problem is a  $|\mathcal{P}|$ -approximation for the MINSUM problem.*

*Proof.* Let  $H$  be a CCQ instance and let  $M^*$  be the optimal matching for the MINMAX problem on  $H$ . For the same instance  $H$ , let  $N^*$  be the optimal matching for the MINSUM problem. Let us define  $t^*$  and  $y^*$  as follows:

$$t^* = \max_{p \in \mathcal{P}} \{c(p) \cdot |M^*(p)|\} \qquad y^* = \sum_{p \in \mathcal{P}} (c(p) \cdot |N^*(p)|)$$

We first observe that  $y^* \geq t^*$ . This is true because  $N^*$  is an  $\mathcal{A}$ -perfect envy-free matching in  $H$ . Furthermore, since costs of all programs are non-negative,  $\max_{p \in \mathcal{P}} \{c(p) \cdot |N^*(p)|\} \leq y^*$ . Therefore if  $y^* < t^*$ , it contradicts the optimality of  $M^*$  for the MINMAX problem. To prove the approximation guarantee, we note that since  $t^*$  denotes the maximum cost incurred at any program in  $M^*$ , the total cost of  $M^*$  is upper bounded by  $|\mathcal{P}| \cdot t^* \leq |\mathcal{P}| \cdot y^*$ .  $\square$

This establishes Theorem 4.2.

## References

1. Santhini K. A., Govind S. Sankar, and Meghana Nasre. Optimal matchings with one-sided preferences: Fixed and cost-based quotas. In *Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems 2022 (to appear)*, 2022.
2. Atila Abdulkadiroğlu and Tayfun Sönmez. School choice: A mechanism design approach. *American Economic Review*, 93(3):729–747, June 2003. doi:10.1257/000282803322157061.
3. Haris Aziz, Anton Baychkov, and Péter Biró. Summer internship matching with funding constraints. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20*, pages 97–104, 2020. URL: <https://dl.acm.org/doi/abs/10.5555/3398761.3398778>.
4. Surender Baswana, Partha Pratim Chakrabarti, Sharat Chandran, Yashodhan Kanoria, and Utkarsh Patange. Centralized admissions for engineering colleges in india. *Interfaces*, 49(5):338–354, 2019. doi:10.1287/inte.2019.1007.
5. Péter Biró, David Manlove, and Shubham Mittal. Size versus stability in the marriage problem. *Theor. Comput. Sci.*, 411(16-18):1828–1841, 2010. doi:10.1016/j.tcs.2010.02.003.
6. Jean Cardinal, Marek Karpinski, Richard Schmied, and Claus Viehmann. Approximating vertex cover in dense hypergraphs. *Journal of Discrete Algorithms*, 13:67 – 77, 2012. doi:<https://doi.org/10.1016/j.jda.2012.01.003>.
7. Frances Cooper. *Fair and large stable matchings in the stable marriage and student-project allocation problems*. PhD thesis, University of Glasgow, UK, 2020. URL: <http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.815027>.
8. Irit Dinur and Shmuel Safra. The importance of being biased. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing*, pages 33–42, 2002. URL: <https://doi.org/10.1145/509907.509915>.
9. Daniel Fragiadakis, Atsushi Iwasaki, Peter Troyan, Suguru Ueda, and Makoto Yokoo. Strategyproof matching with minimum quotas. *ACM Trans. Economics and Comput.*, 4(1):6:1–6:40, 2015. URL: <http://doi.acm.org/10.1145/2841226>, doi:10.1145/2841226.
10. Karthik Gajulapalli, James A. Liu, Tung Mai, and Vijay V. Vazirani. Stability-preserving, time-efficient mechanisms for school choice in two rounds. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020*, volume 182 of *LIPIcs*, pages 21:1–21:15, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.21.
11. D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962. URL: <http://www.jstor.org/stable/2312726>.
12. Dan Gusfield and Robert W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Cambridge, MA, USA, 1989.
13. Chien-Chung Huang and Telikepalli Kavitha. Popular matchings in the stable marriage problem. In *International Colloquium on Automata, Languages, and Programming*, pages 666–677. Springer, 2011. doi:10.1007/978-3-642-22006-7\_56.
14. Rob Irving. Matching practices for entry-labor markets scotland, mip country profile 3. 2011. URL: <https://www.matching-in-practice.eu/the-scottish-foundation-allocation-scheme-sfas/>.
15. Robert W. Irving. Matching medical students to pairs of hospitals: A new variation on a well-known theme. In *Algorithms — ESA' 98*, pages 381–392, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.
16. Robert W. Irving, David F. Manlove, and Gregg O'Malley. Stable marriage with ties and bounded length preference lists. *Journal of Discrete Algorithms*, 7(2):213–219, 2009. Selected papers from the 2nd Algorithms and Complexity in Durham Workshop ACiD 2006. URL: <https://www.sciencedirect.com/science/article/pii/S1570866708000683>, doi:<https://doi.org/10.1016/j.jda.2008.09.003>.
17. Robert W. Irving, David F. Manlove, and Sandy Scott. The stable marriage problem with master preference lists. *Discrete Applied Mathematics*, 156(15):2959–2977, 2008. URL: <https://www.sciencedirect.com/science/article/pii/S0166218X0800022X>, doi:<https://doi.org/10.1016/j.dam.2008.01.002>.
18. Yasushi Kawase and Atsushi Iwasaki. Approximately stable matchings with budget constraints. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18)*, 2018. URL: <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17032>.
19. Prem Krishnaa, Girija Limaye, Meghana Nasre, and Prajakta Nimbhorkar. Envy-freeness and relaxed stability: Hardness and approximation algorithms. In *Algorithmic Game Theory - 13th International Symposium, SAGT 2020*, pages 193–208. Springer, 2020. doi:10.1007/978-3-030-57980-7\_13.
20. A. M. Krishnapriya, Meghana Nasre, Prajakta Nimbhorkar, and Amit Rawat. How good are popular matchings? In *17th International Symposium on Experimental Algorithms, SEA 2018*, pages 9:1–9:14, 2018. doi:10.4230/LIPIcs.SEA.2018.9.
21. Abraham Othman, Tuomas Sandholm, and Eric Budish. Finding approximate competitive equilibria: efficient and fair course allocation. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 873–880, 2010. URL: <https://dl.acm.org/citation.cfm?id=1838323>.
22. Ignacio Rios, Tomas Larroucau, Giorgiogiulio Parra, and Roberto Cominetti. College admissions problem with ties and flexible quotas, 01 2014. doi:10.2139/ssrn.2478998.

23. Paul A Robards. Applying two-sided matching processes to the united states navy enlisted assignment process. Technical report, NAVAL POSTGRADUATE SCHOOL MONTEREY CA, 2001.
24. Alvin E. Roth. On the allocation of residents to rural hospitals: A general property of two-sided matching markets. *Econometrica*, 54(2):425–427, 1986. URL: <http://www.jstor.org/stable/1913160>.
25. Qingyun Wu and Alvin E. Roth. The lattice of envy-free matchings. *Games and Economic Behavior*, 109:201–211, 2018.
26. Wei Yang, JA Giampapa, and Katia Sycara. Two-sided matching for the us navy detailing process with market complication. Technical report, Technical Report CMU-RI-TR-03-49, Robotics Institute, Carnegie-Mellon University, 2003.
27. Yu Yokoi. Envy-free matchings with lower quotas. *Algorithmica*, 82(2):188–211, 2020. URL: <https://doi.org/10.1007/s00453-018-0493-7>, doi:10.1007/s00453-018-0493-7.

## A Omitted details for Section 2

Fig. 6 depicts the key steps of execution of Algorithm 1 on a  $\text{CCQ}_{c_1, c_2}$  instance.

|                                     |                                                                                                                                                                                         |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $a_1 : p_1 \succ p_2 \succ p_0$     | – $M = \{(a_1, p_0), (a_2, p_0)\}$                                                                                                                                                      |
| $a_2 : p_2 \succ p_3 \succ p_0$     | – (1) $a = a_3$ , $y_{a_3} = 1$ , tight edges on $a_3$ are $\{(a_3, p_1), (a_3, p_2), (a_3, p_3)\}$ ,<br>$\text{thresh}(p_1) = \text{thresh}(p_2) = a_1$ and $\text{thresh}(p_3) = a_2$ |
| $a_3 : p_1 \succ p_2 \succ p_3$     | – (3a) $\mathcal{P}(a_3) = \{p_1, p_2, p_3\}$                                                                                                                                           |
| (0) $p_0 : a_1 \succ a_2$           | – (3b,3c) let $a' = a_1$ , then $p = p_2$ , $z_{a_1, p_2, a_3} = 1$                                                                                                                     |
| (1) $p_1 : a_1 \succ a_3$           | – (3d) Tight edges on $a_1$ are $\{(a_1, p_1), (a_1, p_2)\}$ , $p' = p_1$ , $M = \{(a_1, p_1), (a_2, p_0)\}$ , tight edges on $a_3$ are $\{(a_3, p_1), (a_3, p_3)\}$                    |
| (1) $p_2 : a_1 \succ a_2 \succ a_3$ | – (3e) $\text{thresh}(p_1) = a_3$ , $M = \{(a_1, p_1), (a_2, p_0), (a_3, p_1)\}$                                                                                                        |
| (1) $p_3 : a_2 \succ a_3$           | – (3f) $\mathcal{P}(a_3) = \emptyset$                                                                                                                                                   |

**Fig. 6.** A  $\text{CCQ}_{c_1, c_2}$  instance. An execution of Algorithm 1 is illustrated by giving the state of the algorithm. The blue numbers in the bracket correspond to the labels of steps mentioned in the description.

## B Omitted details for Section 3

We first present the omitted proofs.

**Lemma 2** *Before every iteration of the loop starting at line 7, the following holds.*

- (i) *For a matched agent  $\hat{a}$ , either for every program  $p_j \succ_{\hat{a}} M(\hat{a})$ ,  $\text{slack}(\hat{a}, p_j) = c_2 - c_1$  or for every program  $p_j \succ_{\hat{a}} M(\hat{a})$ ,  $\text{slack}(\hat{a}, p_j) = 0$  and  $\text{thresh}(p_j) \neq \hat{a}$ .*
- (ii) *For an unmatched agent  $\hat{a}$ , for every program  $p_j \in \mathcal{N}(\hat{a})$ ,  $\text{slack}(\hat{a}, p_j) = c_2 - c_1$ .*

*Proof.* We will prove this by induction. Before the first iteration of the loop at line 7, from **(P1)** and the initial dual setting, it is clear that the given conditions hold. Suppose the conditions hold before  $l$ -th iteration of the loop. We show that the conditions continue to hold before  $(l + 1)$ -th iteration. During the  $l$ -th iteration, if  $a \neq \hat{a}$  in line 6 and  $a' \neq \hat{a}$  in line 15 then by **Fact 1** and **Fact 2**, none of the dual updates change the slack on any edge incident on  $\hat{a}$ , and  $M(\hat{a})$  also does not change. Thus, in this case the given conditions continue to hold. It is clear that  $a \neq a'$ . Now we consider the case when  $a = \hat{a}$  or  $a' = \hat{a}$  and show that in each of the cases, the conditions hold before  $(l + 1)$ -th iteration begins.

- (i)  **$a = \hat{a}$  in line 6:** It implies that  $\hat{a}$  is unmatched. By induction hypothesis, for every  $p_j \in \mathcal{N}(\hat{a})$ ,  $\text{slack}(\hat{a}, p_j) = c_2 - c_1$ . Then, the update in line 8 makes all edges incident on  $\hat{a}$  tight. We consider the following two cases –  $\hat{a}$  remains unmatched during the  $l$ -th iteration or gets matched.

- **$\hat{a}$  remains unmatched during the  $l$ -th iteration:** Then the while loop at line 14 must have been executed. During an iteration of the loop at line 14, if  $p = p_j$  then the slack of the edge  $(\hat{a}, p_j)$  becomes  $c_2 - c_1$  after the dual update in line 18 (by **Fact 2**). We show that for every  $p_j \in \mathcal{N}(\hat{a})$ , there is some iteration of the loop at line 14 such that  $p = p_j$  is selected, thereby implying that when the loop terminates, for every edge  $(\hat{a}, p_j)$ , slack becomes  $c_2 - c_1$ , hence the given conditions hold.  
Suppose for contradiction that for some program  $p_j$ ,  $p = p_j$  is never selected. Since the edge  $(\hat{a}, p_j)$  is tight before the loop execution began, it must be the case that either  $p_j \prec_{\hat{a}} M(\hat{a})$  or  $\text{thresh}(p_j) = \hat{a}$ . The first case implies that  $M(\hat{a}) \neq \perp$ , a contradiction that  $\hat{a}$  remains unmatched during the  $l$ -th iteration. In the second case, since  $\text{thresh}(p_j) = \hat{a}$ , the edge  $(\hat{a}, p_j)$  was matchable inside the free-promotions routine, thus  $\hat{a}$  must have been matched inside the free-promotions routine, leading to a contradiction again. Thus, for every  $p_j \in \mathcal{N}(\hat{a})$ , there is some iteration of the loop during which  $p = p_j$ .
  - **$\hat{a}$  gets matched during the  $l$ -th iteration:** Recall that all edges incident on  $\hat{a}$  are tight. If  $\hat{a}$  is matched at line 10 then the  $l$ -th iteration immediately terminates. Thus, before the  $(l + 1)$ -th iteration, for every every  $p_j \succ_{\hat{a}} M(\hat{a})$ ,  $\text{slack}(\hat{a}, p_j) = 0$  and by the choice made in line 10,  $\text{thresh}(p_j) \neq \hat{a}$ .  
If  $\hat{a}$  is matched inside the loop at line 14 then the free-promotions routine must have matched it to some program  $p_t$ . We show that in this case, when the loop at line 14 terminates, for every  $p_j \succ_{\hat{a}} p_t$ ,  $\text{slack}(\hat{a}, p_j) = c_2 - c_1$ . First note that when  $p$  is selected at line 17,  $(\hat{a}, p)$  is tight and by **Fact 2**, the update in line 18 changes the slack of edge  $(\hat{a}, p)$  to  $c_2 - c_1$ . When  $\mathcal{P}(\hat{a})$  is re-computed, the program  $p$  is removed from the set hence the slack of  $(\hat{a}, p)$  does not change in subsequent iterations of the loop at line 14. Thus, it is enough to show that for every  $p_j \succ_{\hat{a}} p_t$ ,  $p = p_j$  is chosen in some iteration of the loop at line 14. Suppose not. Then, there exists some  $p_j$  such that  $(\hat{a}, p_j)$  is tight after the loop at line 14 terminates. By the choice of  $p_t$ ,  $(\hat{a}, p_j)$  is not matchable, hence during the last iteration of the loop,  $\mathcal{P}(\hat{a}) \neq \emptyset$  when it was re-computed at line 20, a contradiction that the loop terminates. Therefore, the given conditions are satisfied when the loop at line 14 terminates, that is, before the  $(l + 1)$ -th iteration of the loop at line 7.
- (ii)  **$a' = \hat{a}$  at line 15:** Consider the first iteration of the loop at line 14 when this happens. Since  $\hat{a}$  is a threshold for some program  $p_j \succ_{\hat{a}} M(\hat{a})$ , by the induction hypothesis, for every  $p_j \succ_{\hat{a}} M(\hat{a})$ ,  $\text{slack}(\hat{a}, p_j) = c_2 - c_1$  before this iteration of the loop at line 14. By Corollary 1, when the loop terminates,  $\hat{a}$  is matched such that for every  $p_j \succ_{\hat{a}} M(\hat{a})$ ,  $\text{slack}(\hat{a}, p_j) = 0$  and  $\text{thresh}(p_j) \neq \hat{a}$ . Therefore, the given conditions hold for  $\hat{a}$  before the  $(l + 1)$ -th iteration of the loop at line 7 begins.

□

**Lemma 5** *Matching  $M$  computed by Algorithm 1 is an  $\ell_a$ -approximation of MINSUM.*

Before proving Lemma 5, we show that matched edges are tight. We also show that in an arbitrary iteration of the loop at line 7, at most  $\ell_a$  many distinct  $z$  variables can be updated inside the **else** part of the loop. This bound is used in the charging argument in Lemma 5.

*Claim.* All matched edges are tight.

*Proof.* By observing the initial dual setting and the initial matching, it is clear that all matched edge is tight before line 4. Once the loop at line 7 begins, only a matchable edge is matched, implying that when the edge is tight when matched. By **Fact 2**, an edge  $(a_i, p_j)$  can become slack only because of the update in  $z_{a', p, a}$  variable in line 18. But note that in this step, agent  $a$  is an unmatched agent by choice and hence no edge  $(a_i, p_j)$  incident on a matched agent  $a_i$  can become slack. Thus, all matched edges remain tight at the end of the algorithm. □

*Claim.* Let  $a$  be a fixed unmatched agent selected in line 6 and consider an iteration of the loop at line 7. Suppose the algorithm enters **else** part. Then, at most  $\ell_a$  many distinct  $z_{\times,p,a}$  variables are updated during the execution of the loop at line 14.

*Proof.* For every  $p_j$  selected as  $p$  in line 17, a distinct  $z_{\times,p_j,a}$  variable is updated. Thus, if a fixed program  $p_j$  is selected as  $p$  in line 17 at most once then there are at most  $|\mathcal{P}(a)|$  many distinct  $z_{\times,p,a}$  variables updated during the execution of the loop at line 14. By observing that  $\mathcal{P}(a) \subseteq \mathcal{N}(a) \leq \ell_a$ , the claim follows. Now, we proceed to show that a fixed program  $p_j$  is selected as  $p$  at most once.

Consider the first iteration of the loop at line 14 where  $p = p_j$  is selected. By the choice of  $p_j$ , the edge  $(a, p_j)$  was tight before this iteration. By **Fact 2**, the update on  $z_{\times,p_j,a}$  reduces the lhs of the dual constraint of the edge  $(a, p_j)$  by  $c_2 - c_1$ . Since,  $c_2 > c_1$ , this updates results in making the edge  $(a, p_j)$  slack. No other dual update is performed during the iteration. Therefore, when  $\mathcal{P}(a)$  is re-computed at line 20,  $p_j \notin \mathcal{P}(a)$ . Thus, in a subsequent iteration of this loop,  $p \neq p_j$ .  $\square$

*Proof (of Lemma 5).* Let  $\text{OPT}$  be an optimal matching and  $c(M)$  and  $c(\text{OPT})$  denote the cost of  $M$  and  $\text{OPT}$  respectively. By duality,  $c(\text{OPT}) \geq \sum_{a \in \mathcal{A}} y_a$ .

By Claim B,  $(a, p) \in M$  implies that the edge  $(a, p)$  is tight. Thus, we have

$$\begin{aligned} c(M) &= \sum_{(a,p) \in M} c(p) = \sum_{(a,p) \in M} \left( y_a + \sum_{\substack{p'=p \text{ or } \\ p' \prec_a p}} \sum_{a' \prec_{p'} a} z_{a,p',a'} - \sum_{a \prec_p a'} z_{a',p,a} \right) \\ &= \sum_{a \in \mathcal{A}} y_a + \underbrace{\sum_{(a,p) \in M} \left( \sum_{\substack{p'=p \text{ or } \\ p' \prec_a p}} \sum_{a' \prec_{p'} a} z_{a,p',a'} - \sum_{a \prec_p a'} z_{a',p,a} \right)}_{S(Z)} \end{aligned}$$

where the first equality is from Eq. 1, the second equality is from Eq. 6 and the third equality follows because  $M$  is  $\mathcal{A}$ -perfect. Let  $S(Z)$  denote the second summation in the above cost. Our goal is to show that  $S(Z)$  is upper-bounded by  $(\ell_a - 1) \sum_{a \in \mathcal{A}} y_a$  and thus  $c(M) \leq \ell_a \cdot \sum_{a \in \mathcal{A}} y_a$ .

We first note that all the  $z$  variables are set to 0 initially and they are updated only inside the loop in line 14. We charge the update in every  $z$  variable to a specific unmatched agent  $a$  picked at line 6 and upper-bound the total update in  $z$  charged to  $a$  in terms of  $y_a$ . Let  $A'$  be the set of agents unmatched before the loop at line 5 is entered. These are precisely those agents such that every program in their neighbourhood has cost  $c_2$ . During every iteration of the loop in line 5, an unmatched agent  $a$  from  $A'$  is picked and the loop in line 7 executes until  $a$  is matched. Suppose that after picking  $a$  in line 6, the loop in line 7 runs for  $\kappa(a)$  iterations. Then,  $y_a$  is incremented by  $c_2 - c_1$  for  $\kappa(a)$  times and since  $a$  is matched, it is not picked again at line 6. Thus, at the end of algorithm,  $y_a = c_1 + \kappa(a)(c_2 - c_1)$ , that is  $y_a \geq \kappa(a)(c_2 - c_1)$ .

We first present a simpler analysis that proves an  $(\ell_a + 1)$ -approximation. Recall that the  $z$  variables are non-negative (Eq. 7). Thus, we upper-bound the total value of  $z$  variables appearing in positive form in  $S(Z)$ . During the iterations 1 to  $\kappa(a) - 1$ , the algorithm must enter the **else** part and in the  $\kappa(a)$ -th iteration, the loop may or may not enter the **else** part. Suppose the algorithm enters the **else** part. Then by Claim. B, for a fixed  $a$  when the algorithm enters the **else** part, at most  $\ell_a$  many  $z$  variables are set to  $c_2 - c_1$ . Thus, at most  $\kappa(a)\ell_a(c_2 - c_1)$  total update in  $S(Z)$  occurs during execution of the loop in line 7 when agent  $a$  is picked. We charge this cost to agent  $a$ , thus agent  $a \in A'$  is charged at most  $\ell_a y_a$  in  $S(Z)$ . Thus,

$$c(M) = \sum_{a \in \mathcal{A}} y_a + S(Z) \leq \sum_{a \in \mathcal{A} \setminus A'} y_a + \sum_{a \in A'} y_a + \sum_{a \in A'} \ell_a y_a \leq (\ell_a + 1) \sum_{a \in \mathcal{A}} y_a \leq (\ell_a + 1) c(\text{OPT})$$

Now, we proceed to a better analysis that shows an  $\ell_a$ -approximation. Recall that a  $z$  variable appears in the dual constraint of possibly multiple edges in positive form and in the dual constraint of exactly one edge in negative form. We show that there exist certain  $z$  variables occurring in positive form in the dual constraint of a matched edge that also appear in negative form in the dual constraint of another matched edge, thereby canceling out their contribution in  $S(Z)$ . Thus, it is enough to upper-bound the update in  $z$  variables that are not cancelled. We prove that the total update in such  $z$  variables charged to an agent  $a \in A'$  can be upper-bounded by  $(\ell_a - 1)y_a$  instead of  $\ell_a y_a$  as done earlier.

Suppose  $a$  is matched to  $p_k$  at the end of the algorithm. Then, in the iterations 1 to  $\kappa(a) - 1$ ,  $\text{thresh}(p_k) \neq a$  and the loop in line 14 executes. It implies that in each of the iterations, there exists an agent  $a_j$  such that  $\text{thresh}(p_k) = a_j$  and  $z_{a_j, p_k, a}$  is updated. Also,  $a_j$  was matched to  $p'$  such that  $p' = p_k$  or  $p' \succ_{a_j} p_k$ . By **(P2)**,  $a_j$  never gets demoted, thus at the end of the algorithm,  $M(a_j) = p'$  or  $M(a_j) \succ_{a'} p'$ . Thus, the variable  $z_{a_j, p_k, a}$  appears in positive form in the dual constraint of the edge  $(a_j, M(a_j))$ . Since  $(a, p_k) \in M$  and the variable  $z_{a_j, p_k, a}$  appears in negative form in the dual constraint of edge  $(a, p_k)$ . Therefore, the variable  $z_{a_j, p_k, a}$  cancels out in  $S(Z)$ . This implies that for each of the iterations 1 to  $\kappa(a) - 1$ , at most  $\ell_a - 1$   $z$  variables are set to  $c_2 - c_1$  such that they may not cancel out. We charge the update in these variables to  $a$ .

In the last  $\kappa(a)$ -th iteration,  $a$  gets matched. If  $a$  is matched at line 10 then no  $z$  variable is updated during this iteration. Otherwise,  $a$  is matched in one of the iterations of the loop in line 14 by the free-promotions routine. Suppose, that  $a$  is matched to  $\bar{p}$  in this step. By **(P2)**,  $M(a) = p_k = \bar{p}$  or  $M(a) = p_k \succ_a \bar{p}$ . By the choice of  $\bar{p}$  in the free-promotions routine, the edge  $(a, \bar{p})$  must have been matchable, that is, it is tight and  $\text{thresh}(\bar{p}) = a$ . The fact that edge  $(a, \bar{p})$  was tight implies that no variable of the form  $z_{\times, \bar{p}, a}$  was updated so far during the  $\kappa(a)$ -th iteration. When  $\mathcal{P}(a)$  is re-computed,  $\bar{p} \notin \mathcal{P}(a)$  because  $M(a) = \bar{p}$  at this step. Thus, in the subsequent iterations of the loop in line 14, no agent  $a'$  could have selected  $\bar{p}$  in line 17. This implies that no  $z$  variable of the form  $z_{\times, \bar{p}, a}$  is updated during the rest of the execution of the  $\kappa(a)$ -th iteration. This implies that during the  $\kappa(a)$ -th iteration, the  $z$  variables that are set to  $c_2 - c_1$  are of the form  $z_{\times, \hat{p}, a}$  where  $\hat{p} \in \mathcal{P}(a) \subseteq \mathcal{N}(a)$  and  $\bar{p} \neq \hat{p}$ . By the fact that  $\bar{p} \in \mathcal{N}(a)$  and  $|\mathcal{N}(a)| \leq \ell_a$ , the number such  $z$  variables is at most  $\ell_a - 1$ .

Thus, during  $\kappa(a)$  many iterations for the agent  $a \in A'$  at most  $\kappa(a)(\ell_a - 1)(c_2 - c_1)$  total update in  $S(Z)$  is charged to  $a$ . Recall that  $y_a \geq \kappa(a)(c_2 - c_1)$ . Thus, agent  $a \in A'$  contributes at most  $(\ell_a - 1)y_a$  in  $S(Z)$ . This gives

$$c(M) = \sum_{a \in A} y_a + S(Z) \leq \sum_{a \in A \setminus A'} y_a + \sum_{a \in A'} y_a + \sum_{a \in A'} (\ell_a - 1) \cdot y_a \leq \ell_a \sum_{a \in A} y_a \leq \ell_a \cdot c(\text{OPT})$$

This completes the proof.  $\square$

### B.1 Tightness of the analysis of Algorithm 1

We show that the analysis of our algorithm is tight by presenting a family of  $\text{CCQ}_{c_1, c_2}$  instances for a fixed  $\ell_a$  in Fig. 7. Let  $k \geq 1$ . We have  $\ell_a k + 1$  agents and  $\ell_a + 1$  programs. Program  $p_0$  has cost 0 and rest of the programs have a unit cost. Agent  $a_0$  ranks program  $p_1, \dots, p_{\ell_a}$  in an arbitrary order. Rest of the agents are partitioned into  $\ell_a$  groups of  $k$  each. Each agent  $a_{u,v}$  where  $1 \leq u \leq \ell_a$  and  $1 \leq v \leq k$  ranks program  $p_u$  followed by program  $p_0$ . There exists an arbitrary ordering on agents such that  $a_0$  is the least-preferred agent. In an optimal solution if  $a_0$  is matched to  $p_t$  then  $k$  agents of the form  $a_{t,w}$ ,  $1 \leq w \leq k$  must be matched to  $p_t$  and the rest of the agents are matched to  $p_0$ , therefore the cost of an optimal solution is  $k + 1$ . Algorithm 1 promotes all  $\ell_a k$  agents to their respective program with cost 1, thereby computing a matching with cost  $\ell_a k + 1$ . As  $k$  increases, the approximation guarantee reaches  $\ell_a$ .

$$\begin{array}{ll}
1 \leq u \leq \ell_a, 1 \leq v \leq k, a_{u,v} : p_u \succ p_0 & (0) p_0 : a_{1,1} \succ \dots a_{\ell_a,k} \\
a_0 : p_1 \succ \dots \succ p_{\ell_a} & 1 \leq u \leq \ell_a, (1) p_u : a_{u,1} \succ \dots a_{u,k} \succ a_0
\end{array}$$

**Fig. 7.** A family of instances that illustrate the tightness of analysis of Algorithm 1

## C Omitted details for Section 4.1

In Fig. 8, we present a family of instances with  $n$  agents and 3 programs where  $c(p_0) = 0$ ,  $c(p_1) = 1$ ,  $c(p_2) = n$  and  $\ell_p = n$ . The lower-bound  $lb_1$  is 1 and an optimal matching in this instance has cost  $n$ . Thus the best approximation guarantee using  $lb_1$  as the lower-bound is  $\ell_p$ . We remark that this holds even under master list ordering on agents and programs.

$$\begin{array}{ll}
1 \leq i \leq n-1, & a_i : p_1 \succ p_0 \\
a_n : p_1 \succ p_2 & \\
(0) p_0 : a_1 \succ a_2 \succ \dots \succ a_{n-1} & \\
(1) p_1 : a_1 \succ a_2 \succ \dots \succ a_{n-1} \succ a_n & \\
(n) p_2 : a_n &
\end{array}$$

**Fig. 8.** A family of instances with optimal cost exactly  $\ell_p$  times the lower-bound  $lb_1$ . There are two optimal matchings of cost  $n$ :  $\text{OPT}_1 = \{(a_1, p_0), \dots, (a_{n-1}, p_0), (a_n, p_2)\}$  and  $\text{OPT}_2 = \{(a_1, p_1), \dots, (a_n, p_1)\}$ .

## D Proof of Theorem 2

In this section, we prove the NP-hardness and inapproximability result (Theorem 2) for the MINSUM problem.

### D.1 NP-hardness of MINSUM

We show that the MINSUM problem is NP-hard even under severe restrictions on the instance. In particular we show that the hardness holds even when all agents have a preference list of a constant length  $f$  and the instance has two distinct costs and there is a master list ordering on agents and programs.

**Reduction:** Let  $\langle S, E, k \rangle$  be an instance of Set Cover (SC) such that every element in  $E$  occurs in exactly  $f$  sets. Minimum vertex cover on  $f$ -uniform hypergraphs is known to be NP-complete and SC problem where every element occurs in exactly  $f$  sets is equivalent to it [6]. Let  $m = |S|$ ,  $n = |E|$ . We construct an instance  $H$  of CCQ as follows. For every set  $s_i \in S$ , we have a set-agent  $a_i$  and a set-program  $p_i$ . For every element  $e_h \in E$ , we have an element-agent  $a'_h$ . We also have a program  $p$  and  $f - 2$  programs  $p^1, \dots, p^{f-2}$ . Thus in the instance  $H$  we have  $m + n$  agents and  $m + f - 1$  programs. Let  $E_i$  denote the set of elements in the  $s_i$ . The element-agents corresponding to  $E_i$  in  $H$  are denoted by  $\mathcal{A}'_i$ .

**Preferences:** The preference lists of agents and programs are shown in Fig. 9. Every set-agent  $a_i$  has  $f$  programs in its preference list – the set-program  $p_i$  followed by the program  $p$  followed by the programs  $p^1, \dots, p^{f-2}$  in that order. Every element-agent  $a'_j$  has the  $f$  set-programs corresponding to the sets that contain it in an arbitrary fixed order. Every set-program  $p_i$  has its set-agent  $a_i$  as its top-preferred agent followed by the agents in  $\mathcal{A}'_i$  in an arbitrary fixed order. The program  $p$  and each program  $p^1, \dots, p^{f-2}$  has the set-agents as  $a_1, \dots, a_n$  in an arbitrary fixed order. It is clear that every agent has a preference list of length  $f$ .

**Costs:** The costs for program  $p_i$  for  $1 \leq i \leq n$  and  $p^j$  for  $1 \leq j \leq f - 2$  is 1 and that for the program  $p$  is 0. Thus, the instance has two distinct costs.

**Lemma 9.**  $\langle S, E, k \rangle$  is a yes instance if and only if  $H$  admits an  $\mathcal{A}$ -perfect envy-free matching of cost at most  $n + k$ .

$$\begin{array}{ll}
 1 \leq i \leq m, a_i : p_i \succ p \succ p^1 \succ \dots \succ p^{f-2} & 1 \leq i \leq m, p_i : a_i \succ \mathcal{A}'_i \\
 1 \leq j \leq n, a'_j : p_{j1} \succ \dots \succ p_{jf} & p : a_1 \succ \dots \succ a_m \\
 & 1 \leq t \leq f-2, p^t : a_1 \succ \dots \succ a_m
 \end{array}$$

**Fig. 9.** Reduced instance  $H$  of CCQ from instance  $\langle S, E, k \rangle$  of SC

*Proof.* Let  $X \subseteq S$  be the set cover size at most  $k$ . Using  $X$ , we construct an  $\mathcal{A}$ -perfect matching  $M$  in  $H$  and show that  $M$  is envy-free. We then show that the cost of  $M$  is bounded. For every set  $s_i \notin X$ , we match the corresponding set-agent  $a_i$  to the program  $p$ , that is,  $M(a_i) = p$ . For every set  $s_i \in X$ , we match the corresponding set-agent  $a_i$  to the program  $p_i$ , that is,  $M(a_i) = p_i$ . Since  $X$  is a set cover, for every element  $e_j \in E$ , at least one of the sets it occurs in is in  $X$ . Thus, for every element-agent  $a'_j$ , we match it to the program  $p_{jt}$  corresponding to the set in  $X$  (in case more than one set containing  $e_j$  are in  $X$ , then we match it to the program which is highest-preferred among them). It is clear that  $M$  is  $\mathcal{A}$ -perfect.

To prove that  $M$  is envy-free, we show that no agent participates in an envy-pair. First observe that, an agent  $a_i$  corresponding to  $s_i \in X$  is matched to its top-choice program. Hence such agents do not participate in envy pairs. Now, for all agents  $a_i$  such that  $s_i \notin X$ , these agents are matched to  $p$ . However, the corresponding program  $p_i$  remains closed. Finally, every element-agent that is not matched to its top-choice programs has all such top-choice programs closed. Thus the matching is envy-free. We compute the cost of matching  $M$  from the agent side. Each element-agent is matched to some program  $p_i$  and costs 1 each. At most  $k$  set-agents matched to their corresponding set-program  $p_i$  each costs 1 and the  $m - k$  set-agents are matched to program  $p$  that incur the cost 0 each. Hence the cost of the matching is at most  $n + k$ .

Now assume that  $H$  admits an  $\mathcal{A}$ -perfect envy-free matching of size  $n + k$ . First we prove that program  $p$  must take at least  $m - k$  set-agents - Since the matching is  $\mathcal{A}$ -perfect, every matched element-agent contributes a cost of at least 1 each, every set-agent not matched to  $p$  contributes a cost of 1 each thus if program  $p$  takes less than  $m - k$  set-agents then the cost of any such  $\mathcal{A}$ -perfect envy-free matching is at least  $n + k + 1$ , which is a contradiction. Thus, at least  $m - k$  set-agents must be matched to program  $p$ .

Let  $X$  be the set of sets  $s_i$  such that  $a_i$  is matched to  $p_i$ . Then,  $|X| \leq k$ . Since, the matching is envy-free, every program  $p_i$  such that  $s_i \notin X$  must be closed (since  $a_i$  is either matched to  $p$  or  $p^j$ ). We now prove that  $X$  is a set cover. Suppose not, then there exists at least one element, say  $e_j$  such that no set containing  $e_j$  is in  $X$ , implying that all the programs  $p_{jt}$  are closed. It means that element-agent  $a'_j$  is unmatched. This implies that the matching is not  $\mathcal{A}$ -perfect, which is a contradiction. Hence  $X$  must be a set cover. Since  $|X| \leq k$ , thus  $\langle S, E, k \rangle$  is a yes instance.  $\square$

Lemma 9 shows that the reduction is correct. We note that the costs in the reduced instance of CCQ are constant, thus MINSUM is strongly NP-hard. We remark that the NP-hardness result holds even when there is a master list ordering on agents and programs as follows:  $(a_1 \succ \dots \succ a_m \succ a'_1 \succ \dots \succ a'_n)$ ,  $(p_1 \succ \dots \succ p_m \succ p \succ p^1 \succ \dots \succ p^{f-2})$ . This establishes Theorem 2.1.

## D.2 Inapproximability of MINSUM

We show a reduction from the Minimum Vertex Cover problem.

**Reduction:** Let  $G = (V, E)$  be an instance of Minimum Vertex Cover (MVC) problem. We construct an instance of CCQ as follows. Let  $n = |V|$  and  $m = |E|$ . For every vertex  $u_i$ , we have  $m$  vertex-agents  $a_i^1, \dots, a_i^m$  and 2 vertex-programs  $p_i$  and  $p'_i$ . For every edge  $e_j$ , we have one edge-agent  $a'_j$ . We have an additional program  $p$ . Thus, we have  $m + mn$  agents and  $2n + 1$  programs.

**Preferences:** Let  $\mathcal{P}'_j$  denote the set of vertex-programs  $p'_{j1}$  and  $p'_{j2}$  corresponding to the end-points  $u_{j1}$  and  $u_{j2}$  of edge  $e_j$  and  $\mathcal{A}'_i$  denote The set of edge-agents  $a'_j$  corresponding to the edges incident on vertex  $u_i$ . Each vertex-agent  $a_i^t, 1 \leq t \leq m$  has the programs  $p_i, p'_i, p$  in that order. Each edge-agent  $a'_j$  has the two programs in  $\mathcal{P}'_j$  in an arbitrary but fixed order. Each vertex-program  $p_i$  has the  $m$  vertex-agents  $a_i^1, \dots, a_i^m$  in that order. Each vertex-program  $p'_i$  has the  $m$  vertex-agents  $a_i^1, \dots, a_i^m$  in that order, followed by the edge-agents  $\mathcal{A}'_i$ . Program  $p$  has the  $mn$  vertex-agents in an arbitrary, fixed order.

**Costs:** The cost of each vertex-program  $p_i$  is 3, that of each vertex-program  $p'_i$  is  $2n$  and that of the program  $p$  is 0. Note that  $2n > 3$  since  $n \geq 2$ .

$$\begin{aligned} \forall 1 \leq i \leq n, 1 \leq j \leq m, \quad a_i^j : p_i \succ p'_i \succ p & \quad \forall 1 \leq i \leq n, \quad p_i : a_i^1 \succ \dots \succ a_i^m \\ 1 \leq j \leq m, \quad a'_j : \mathcal{P}'_j & \quad \forall 1 \leq i \leq n, \quad p'_i : a_i^1 \succ \dots \succ a_i^m \succ \mathcal{A}'_i \\ & \quad 1 \leq i \leq n, 1 \leq t \leq m, \quad p : a_i^t \end{aligned}$$

**Fig. 10.** Reduced instance  $H$  of CCQ from instance  $G$  of MVC

**Lemma 10.** *If an optimal vertex cover in  $G$  is of size at most  $(\frac{2}{3} + \epsilon) \cdot n$  then in  $H$ , there exists an  $\mathcal{A}$ -perfect envy-free matching  $M$  with cost at most  $(4 + 3\epsilon)mn$ .*

*Proof.* Let  $V'$  be the vertex cover of  $G$  of size at most  $(\frac{2}{3} + \epsilon) \cdot n$ . If  $u_i \notin V'$ , we match all the  $m$   $a_i^t$  agents to the program  $p$  at cost 0. If  $u_i \in V'$ , we match all the  $m$   $a_i^t$ -type agents to the program  $p_i$ . This contributes the cost of at most  $m \cdot (\frac{2}{3} + \epsilon) \cdot n \cdot 3 = mn(2 + 3\epsilon)$ . We then prune the preference list of every edge-agent  $a'_j$  by deleting the programs  $p'_i$  corresponding to the end-point  $u_i$  such that  $u_i \notin V'$ . Since  $V'$  is a vertex cover, it is guaranteed that every edge is covered and hence every edge-agent  $a'_j$  has a non-empty list  $\mathcal{P}'_j$  after pruning. Every edge-agent  $a'_j$  is then matched to the top-preferred program  $p'_i$  in the pruned list  $\mathcal{P}'_j$ . This contributes a cost of  $2mn$  incurred by the edge-agents. Thus, the total cost of this matching is at most  $(2 + 2 + 3\epsilon)mn = (4 + 3\epsilon)mn$ . It is easy to see that  $M$  is  $\mathcal{A}$ -perfect and is envy-free since no agent participates in an envy-pair.  $\square$

**Lemma 11.** *If the optimal vertex cover in  $G$  has size greater than  $(\frac{8}{9} - \epsilon) \cdot n$  then in  $H$ , any  $\mathcal{A}$ -perfect envy-free matching  $M$  has cost greater than  $(\frac{14}{3} - 3\epsilon)mn$ .*

*Proof.* We prove the contra-positive i.e. if there exists an  $\mathcal{A}$ -perfect envy-free matching within cost  $mn(\frac{14}{3} - 3\epsilon)$  then optimal vertex cover in  $G$  has size at most  $(\frac{8}{9} - \epsilon) \cdot n$ . Given that the matching is  $\mathcal{A}$ -perfect, all the  $a'_j$  type agents ( $m$  in total) must be matched to some program in  $\mathcal{P}'_j$ . Note that each such agent must contribute a cost of  $2n$  and hence they together contribute a cost of  $2mn$ .

Suppose the edge-agent  $a'_j$  is matched to a program  $p'_i$  such that  $u_i$  is one of the end-points of edge  $e_j$ . Then,  $m$   $a_i^t$  vertex-agents for the vertex  $u_i$  must be matched to either  $p_i$  or  $p'_i$ , otherwise they form an envy pair with  $a'_j$ . Let  $V'$  be the set of vertices  $u_i$  such that at least one  $a_i^t$  is matched to  $p_i$  or  $p'_i$ . If  $u_i \in V'$  is such that less than  $m$   $a_i^t$  type agents are matched to  $p_i$  or  $p'_i$ , then it implies that no edge-agent corresponding to the edge incident on  $u_i$  could have been matched to  $p'_i$  but is matched to  $p'_k$  where  $u_k$  is its other end-point. Thus, we can remove such  $u_i$  from  $V'$ . It is clear that  $V'$  is a vertex cover of  $G$  since the matching is  $\mathcal{A}$ -perfect. For every  $u_i \in V'$ , all  $m$   $a_i^t$  vertex-agents are matched to  $p_i$  or  $p'_i$ . Since they together contribute the cost at most  $mn(\frac{8}{3} - 3\epsilon)$  and each  $u_i \in V'$  has  $m$  copies matched, each contributing a cost of at least 3, it implies that  $|V'| \leq (\frac{8}{9} - \epsilon) \cdot n$ , implying that an optimal vertex cover in  $G$  has size at most  $(\frac{8}{9} - \epsilon) \cdot n$ .  $\square$

**Lemma 12.** *The MINSUM problem is NP-hard to approximate within a factor of  $\frac{7}{6} - \delta$  for any constant  $\delta > 0$ , unless  $P = NP$ .*

*Proof.* We use the following proposition and the results proved in Lemma 10 and Lemma 11.

**Proposition [8].** For any  $\epsilon > 0$  and  $p < \frac{3-\sqrt{5}}{2}$ , the following holds: If there is a polynomial-time algorithm that, given a graph  $G = (V, E)$ , distinguishes between the following two cases, then  $P = NP$ .

- (1)  $|VC(G)| \leq (1 - p + \epsilon)|V|$
- (2)  $|VC(G)| > (1 - \max\{p^2, 4p^3 - 3p^4\} - \epsilon)|V|$   $\square$

By letting  $p = \frac{1}{3}$  in Proposition above, we know that the existence of a polynomial-time algorithm that distinguishes between the following two cases implies  $P = NP$  for an arbitrary small positive constant  $\epsilon$ :

- (1)  $|VC(G)| \leq (\frac{2}{3} + \epsilon)|V|$  i.e.  $\text{cost}(M) \leq (4 + 3\epsilon)mn$
- (2)  $|VC(G)| > (\frac{8}{9} - \epsilon)|V|$  i.e.  $\text{cost}(M) > (\frac{14}{3} - 3\epsilon)mn$

Now, suppose that there is a polynomial-time approximation algorithm  $A$  for MINSUM whose approximation ratio is at most  $\frac{7}{6} - \delta$  for some  $\delta$ . Then, consider a fixed constant  $\epsilon$  such that  $\epsilon < \frac{8\delta}{13-6\delta}$ . If an instance of case (1) is given to  $A$ , it outputs a solution with cost at most  $(4 + 3\epsilon)mn(\frac{7}{6} - \delta) < mn\frac{26(7-6\delta)}{3(13-6\delta)}$ . If an instance of case (2) is given to  $A$ , it outputs a solution with cost greater than  $mn(\frac{14}{3} - 3\epsilon) > mn\frac{26(7-6\delta)}{3(13-6\delta)}$ . Hence, using  $A$ , we can distinguish between cases (1) and (2), which implies that  $P = NP$ .  $\square$

This establishes Theorem 2.2.