Dynamic Rank-Maximal Matchings

Prajakta Nimbhorkar[1] and Arvind Rameshwar V.[2*]

[1] Chennai Mathematical Institute (`prajakta@cmi.ac.in`)
[2] Birla Institute of Technology and Science, Hyderabad Campus
(`arvind.rameshwar@gmail.com`)

Abstract. We consider the problem of matching applicants to posts where applicants have preferences over posts. Thus the input to our problem is a bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, E)$, where $\mathcal{A}$ denotes a set of applicants, $\mathcal{P}$ is a set of posts, and there are ranks on edges which denote the preferences of applicants over posts. A matching M in G is called *rank-maximal* if it matches the maximum number of applicants to their rank 1 posts, subject to this the maximum number of applicants to their rank 2 posts, and so on.

We consider this problem in a dynamic setting, where vertices and edges can be added and deleted at any point. Let n and m be the number of vertices and edges in an instance G, and r be the maximum rank used by any rank-maximal matching in G. We give a simple $O(r(m+n))$-time algorithm to update an existing rank-maximal matching under each of these changes. When $r = o(n)$, this is faster than recomputing a rank-maximal matching completely using a known algorithm like that of Irving et al. [13], which takes time $O(\min((r + n, r\sqrt{n})m)$.

1 Introduction

We consider matchings under one-sided preferences. The problem can be modeled as that of matching applicants to posts where applicants have preferences over posts. This problem has several important practical applications like allocation of graduates to training positions [11] and families to government housing [18]. The input to the problem consists of a bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, E)$, where $\mathcal{A}$ is a set of applicants, $\mathcal{P}$ is a set of posts. Each applicant has a subset of posts ranked in an order of preference. This is referred to as the *preference list* of the applicant. An edge (a, p) has rank i if p is an ith choice of a. An applicant can have any number of posts at rank i, including zero. Thus the edge-set E can be partitioned as $E = E_1 \dot\cup \ldots \dot\cup E_r$, where E_i contains the edges of rank i.

This problem has received lot of attention and there exist several notions of optimality like pareto-optimality [1], rank-maximality [13], popularity [2], and fairness. The notion of *rank-maximality* has been first studied by Irving [12], who called it *greedy matchings* and also gave an algorithm for computing such matchings in case of strict lists. A rank-maximal matching matches maximum

---

number of applicants to their rank 1 posts, subject to that, maximum number of applicants to their rank 2 posts and so on. Irving et al.[13] gave an $O(\min(n+r, r\sqrt{n})m)$-time algorithm to compute a rank-maximal matching. Here $n = |\mathcal{A}| + |\mathcal{P}|$, $m = |E|$, and r denotes the maximum rank on any edge in a rank-maximal matching. The weighted and capacitated versions of this problem have been studied in [14] and [16] respectively.

We consider the rank-maximal matching problem in a dynamic setting where vertices and edges are added and deleted over time. The requirement of dynamic updates in matchings has been well-studied in literature, with the motivation of updating an existing optimal matching without recomputing it completely. Dynamic updates are important in real-world applications as applicants matched to posts can leave their jobs, or new applicants can apply for a job, or an applicant can acquire new skills and hence becomes eligible for more posts.

Related work: Bipartite matchings as well as popular matchings have been extensively studied in a dynamic setting [15,4,10,5,6] [7,3]. The algorithms for maintaining maximum matchings in dynamic bipartite graphs maintain a matching under addition and deletion of edges that closely approximates the maximum cardinality matching, and the update time is small i.e. sub-linear or even poly-logarithmic in the size of the graph. The algorithm of [7] maintains a matching that has an unpopularity factor of $(\Delta+k)$ with $O(\Delta+\Delta^2/k)$ amortized changes per round for addition or deletion of an edge, and $O(\Delta^2+\Delta^3/k)$ changes per round for addition and deletion of a vertex for any $k > 0$. In contrast to this, our algorithm maintains rank-maximal matchings exactly but needs $O(r(m+n))$ time for each update. We describe our contribution below.

Recently, independent of our work, [8] give an $O(m)$ algorithm for updating rank-maximal matchings under addition and deletion of vertices using techniques similar to ours.

1.1 Our Contribution

We consider the problem of updating an existing rank-maximal matching when a vertex or edge is added or deleted. We show the following in this paper:

Theorem 1. *Given an instance of the rank-maximal matching problem with n vertices and m edges, there is an $O(r(m + n))$-time algorithm for updating a rank-maximal matching when a vertex or edge is added to or deleted from the instance. Here r is the maximum rank used in any rank-maximal matching in the instance.*

When $r = o(n)$, this is faster than recomputing a rank-maximal matching using the fastest known algorithm by Irving et al.[13].

Our algorithm crucially uses Irving et al.'s algorithm and the graphs it creates for each stage. In Irving et al.'s algorithm, at stage i, edges of rank i are added to the instance and some edges which can not belong to any rank-maximal matching are deleted. We show that addition or deletion of a vertex or edge can lead to addition and deletion of several edges at each stage, however, at most one augmenting path is created at each stage. This helps us update each stage

in time $O(m+n)$, thus total time taken is $O(r(m+n))$ where r is the maximum rank on any edge in a rank-maximal matching.

It is important to note that addition or deletion of even one edge can change an existing rank-maximal matching by as much as $\Omega(n)$ edges. We give an example in Appendix to show this. Also, addition or deletion of a vertex can potentially lead to addition or deletion of $\Omega(n)$ edges. In light of this, it is an interesting aspect of our algorithm that it avoids a complete recomputation of a rank-maximal matching. Also, in the instances that arise in practice, where there is a large number of applicants and posts, typically each applicant ranks only a small subset of posts. Therefore our algorithm is useful for updating a rank-maximal matching in such instances substantially faster than recomputing it completely.

1.2 Organization of the paper

In Section 2, we give some definitions and recall the algorithm of Irving et al.[13] for computing a rank-maximal matching along with some of its properties. The preprocessing and an overview of the algorithm appear in Section 3. The description and analysis of the algorithm is given in Section 4. We discuss some related questions in Section 5.

2 Preliminaries

We recall some well-known definitions and terminology (see e.g. [9]). A matching M in a graph G is a subset of edges, such that no two of them share a vertex. For a matched vertex u, we denote by $M(u)$ its partner in M.

Properties of maximum matchings in bipartite graphs: Let $G = (\mathcal{A} \cup \mathcal{P}, E)$ be a bipartite graph and let M be a maximum matching in G. The matching M defines a partition of the vertex set $\mathcal{A} \cup \mathcal{P}$ into three disjoint sets, defined below:

Definition 1 (Even, odd, unreachable vertices). *A vertex $v \in \mathcal{A} \cup \mathcal{P}$ is* even *(resp.* odd*) if there is an even (resp. odd) length alternating path with respect to M from an unmatched vertex to v. A vertex v is* unreachable *if there is no alternating path from an unmatched vertex to v.*

The following lemma is well-known in matching theory; see [17] or [13] for a proof.

Lemma 1 ([17]). *Let $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ be the sets of even, odd, and unreachable vertices defined by a maximum matching M in G. Then,*

(a) $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ are disjoint, and are the same for all the maximum matchings in G.

3

(b) In any maximum matching of G, every vertex in $\mathcal{O}$ is matched with a vertex in $\mathcal{E}$, and every vertex in $\mathcal{U}$ is matched with another vertex in $\mathcal{U}$. The size of a maximum matching is $|\mathcal{O}| + |\mathcal{U}|/2$.

(c) No maximum matching of G contains an edge with one end-point in $\mathcal{O}$ and the other in $\mathcal{O} \cup \mathcal{U}$. Also, G contains no edge with one end-point in $\mathcal{E}$ and the other in $\mathcal{E} \cup \mathcal{U}$.

Rank-maximal matchings: An instance of the rank-maximal matchings problem consists of a bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, E)$, where $\mathcal{A}$ is a set of applicants, $\mathcal{P}$ is a set of posts, and applicants rank posts in order of their preference. That is the input is a bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, E)$ where the edges in E can be partitioned as $E_1 \cup E_2 \cup \ldots \cup E_r$. Here E_i denotes the edges of rank i, and r denotes the maximum rank any applicant assigns to a post. An edge (a, p) has rank i if p is an ith choice of a.

Definition 2 (Signature). *The* signature *of a matching M is defined as an r-tuple $\rho(M) = (x_1, \ldots, x_r)$ where, for each $1 \leq i \leq r$, x_i is the number of applicants who are matched to their ith rank post in M.*

Let M, M' be two matchings in G, with signatures $\rho(M) = (x_1, \ldots, x_r)$ and $\rho(M') = (y_1, \ldots, y_r)$. Define $M \succ M'$ if $x_i = y_i$ for $1 \leq i < k \leq r$ and $x_k > y_k$.

Definition 3 (Rank-maximal matching). *A matching M in G is rank-maximal if M has the maximum signature under the above ordering $\succ$.*

Observe that all the rank-maximal matchings in an instance have the same cardinality and the same signature.

Construction of Rank-maximal Matchings: Now we recall Irving et al.'s algorithm [13] for computing a rank-maximal matching in a given instance $G = (\mathcal{A} \cup \mathcal{P}, E_1 \cup \ldots \cup E_r)$. The pseudocode of the same appears in Algorithm 1 and a description is given below. Recall that E_i is the set of edges of rank i.

Let $G_i = (\mathcal{A} \cup \mathcal{P}, E_1 \cup \ldots \cup E_i)$. The algorithm involves r stages, each stage i considers edges of rank at most i. The algorithm starts with $G'_1 = G_1$. A maximum matching M_1 is computed in G_1. Then the vertices are labelled as even, odd, unreachable with respect to M_1. These sets of vertices are called $\mathcal{E}_1, \mathcal{O}_1, \mathcal{U}_1$ respectively. The edges between two vertices in $\mathcal{O}_1$ or a vertex in $\mathcal{O}_1$ and another in $\mathcal{U}_1$ can not belong to any maximum matching, and hence they are deleted. Moreover, all the vertices in $\mathcal{O}_1 \cup \mathcal{U}_1$ have to be matched by any maximum matching in G_1. Therefore edges of rank more than 1 incident on such vertices are also deleted from G. The resulting graph is called *the reduced graph* G'_1. The same process is repeated for each stage. Thus, at stage i, edges of rank i are added to G'_{i-1} i.e. the reduced graph of stage $i-1$ to get G'_i. A maximum matching M_i is computed in G'_i by augmenting the matching M_{i-1} from the previous stage, and the vertices are partitioned into sets $\mathcal{E}_i, \mathcal{O}_i, \mathcal{U}_i$. Then the edges between two vertices in $\mathcal{O}_i$ or between a vertex in $\mathcal{O}_i$ and a vertex in $\mathcal{U}_i$

are deleted. Edges of rank more than i incident on the vertices in $\mathcal{O}_i \cup \mathcal{U}_i$ are also deleted from G. This is the reduced graph of stage i, called G'_i. It is shown in [13] that the matching M_i is rank-maximal in G_i.

Algorithm 1 An algorithm to compute a rank-maximal matching from [13].

Input: $G = (\mathcal{A} \cup \mathcal{P}, E_1 \cup E_2 \cup \cdots \cup E_r)$.
Output: A rank maximal matching M in G.
 1: Let $G_i = (\mathcal{A} \cup \mathcal{P}, E_1 \cup E_2 \cup \cdots \cup E_i)$
 2: Construct $G'_1 = G_1$. Let M_1 be a maximum matching in G'_1.
 3: **for** $i = 1 \ldots r$ **do**
 4:     Partition $\mathcal{A} \cup \mathcal{P}$ as $\mathcal{O}_i, \mathcal{E}_i, \mathcal{U}_i$ with respect to M_i in G'_i.
 5:     Delete all edges of rank $j > i$ incident on vertices in $\mathcal{O}_i \cup \mathcal{U}_i$.
 6:     Delete all edges from G'_i between a node in $\mathcal{O}_i$ and a node in $\mathcal{O}_i \cup \mathcal{U}_i$.
 7:     Add edges in E_{i+1} to G'_i; denote the resulting graph G'_{i+1}.
 8:     Compute a maximum matching M_{i+1} in G_{i+1} by augmenting M_i.
 9: **end for**
10: Delete all edges from G'_{r+1} between a node in $\mathcal{O}_{r+1}$ and a node in $\mathcal{U}_{r+1}$.
11: Denote the graph G'_{r+1} as G'.
12: Return a rank-maximal matching $M = M_{r+1}$.

We note the following properties of Irving et al.'s algorithm:

(*I*1) For every $1 \le i \le r$, every rank-maximal matching in G_i is contained in G'_i.
(*I*2) The matching M_i is rank-maximal in G_i, and is a maximum matching in G'_i.
(*I*3) If a rank-maximal matching in G has signature $(s_1, \ldots, s_i, \ldots s_r)$ then M_i has signature $(s_1, \ldots, s_i)$.
(*I*4) The graphs G'_i, $1 \le i \le r$ constructed at the end of iteration i of Irving et al.'s algorithm, and G' are independent of the rank-maximal matching computed by the algorithm. This follows from Lemma 1 and invariant *I*3.

3   Preprocessing and overview

In the preprocessing stage, we store the information necessary to perform an update in $O(r(m + n))$ time. The preprocessing time is asymptotically same as that of computing a rank-maximal matching in a given instance by Irving et al.'s algorithm viz. $O(\min((r + n, r\sqrt{n})m))$ and uses $O((m + n)\log n)$ storage.

3.1   Preprocessing

Given an instance of the rank-maximal matching problem, $G = (\mathcal{A} \cup \mathcal{P}, E)$ and ranks on edges, we execute Irving et al.'s algorithm on G. (Algorithm 1 from Section 2.) Recall that n is the number of vertices and m is the number of edges.

We use the reduced graphs G'_i for $1 \le i \le r$, where $G'_i = (\mathcal{A} \cup \mathcal{P}, E'_i)$, computed by Algorithm 1 for updating a rank-maximal matching in G on addition

or deletion of an edge or a vertex. If M is a rank-maximal matching in G, then in each G_i', we consider the matching $M_i = M \cap E_i'$. By Invariant $(I2)$ from Section 2, M_i is rank-maximal in G_i. When a vertex or an edge is added to or deleted from G, the goal is to emulate Algorithm 1 on the new instance H using the reduced graphs G_i' for each i.

We prove in Lemma 2 below that we do not need to store the reduced graphs explicitly. The storage can be achieved by storing the original graph G along with some extra information for each stage. If a vertex becomes odd (respectively unreachable) at stage i of Algorithm 1, we store the number i and one bit 0 (respectively 1) indicating that, at stage i, the vertex became odd (respectively unreachable). For each edge, we store the stage at which it gets deleted, if at all. This takes $O((m+n)\log n)$ extra storage.

Lemma 2. *A reduced graph G_i' of any stage i of Algorithm 1 can be completely reconstructed from the stored information as described above. Moreover, this reconstruction can be done in $O(m+n)$ time.*

Proof. Edge-set E_i' of G_i' is a subset of $E_1 \cup \ldots \cup E_i$. We go over all the edges in $E_1 \cup \ldots \cup E_i$ and keep those edges in E_i' which have not been deleted up to stage i. This is precisely the information we have stored for each edge. As we go over each edge exactly once, we need $O(m+n)$ time.

3.2 An overview of the algorithm

Let G be a given instance and let H be the updated instance obtained by addition or deletion of an edge or a vertex. As stated earlier, the goal of our algorithm is to emulate Algorithm 1 on H using stored in the preprocessing step described above. Thus our algorithm constructs the reduced graphs H_i' for H by updating the reduced graphs G_i', and also a rank-maximal matching M' in H by updating a rank-maximal matching M in G. We prove that the graphs H_i' are same as the reduced graphs that would be obtained by executing Algorithm 1 on H.

The reduced graph H_i' can be significantly different from the reduced graph G_i' for a stage i. However, we show that there is at most one augmenting path in H_i' for any stage i. Thus each H_{i+1}' and M' can be obtained from H_i', G_{i+1}', and M_i in linear time i.e. $O(m+n)$ time.

We note that, in Irving et al.'s algorithm, an applicant is allowed to have any number of posts of a rank i, including zero. Also, because of the one-sided preferences model, each edge has a unique rank associated with it. Thus addition of an applicant is analogous to addition of a post. In both the cases, a new vertex is added to the instance, along with the edges incident on it, and along with the ranks on these edges. The ranks can be viewed from either applicants' side or posts' side. Therefore, we describe our algorithm for addition of an applicant, but the same can be used for addition of a post. The same is true for deletion of a vertex. Deletion of an applicant or post involves deleting a vertex, along with its incident edges. Hence the same algorithm applies to deletion of both applicants and posts.

4 The Algorithm

We describe the update algorithm here. Throughout this discussion, we assume that G is an instance of rank-maximal matchings and H is an updated instance, where an update could be addition or deletion of an edge or a vertex. We discuss each of these updates separately.

As described in Section 3, we first run Algorithm 1 on G and compute a rank-maximal matching M in G. We also store the information regarding each vertex and edge as described in Section 3. In the subsequent discussion, we assume that we have the reduced graphs G'_i for each rank $1 \leq i \leq r$, which can be obtained in linear-time from the stored information as proved in Lemma 2.

4.1 Addition of a vertex:

We describe the procedure for addition of a vertex in terms of addition of an applicant. Addition of a post is analogous as explained in Section 3. A description of the vertex-addition algorithm is given below and then we prove its correctness. The pseudocode is given in Appendix.

Description of vertex-addition algorithm: Let a be a new applicant to be added to the instance G. Let E_a be the set of edges along with their ranks, that correspond to the preference list of a. Thus the new instance is $H = ((\mathcal{A} \cup \{a\}) \cup \mathcal{P}, E \cup E_a)$. The update algorithm starts from G'_1, adds edges of rank 1 from E_a to G'_1 to get H'_1 and then updates M and H'_1 as follows:

Initialization: $S, T = \emptyset$. These sets are used later as described below.

The following cases arise while updating H'_1:

Case 1: Each rank 1 post p of a is odd in G'_1: Then H'_1 is same as G'_1, along with a and its rank 1 edges added.

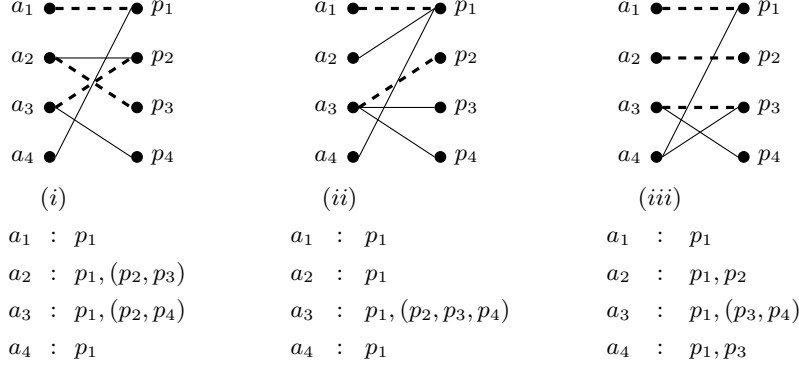**Case 2: No rank 1 post of a is even but some post is unreachable in G'_1:** Update the labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$.[3] Add those applicants whose label changes from $\mathcal{U}$ to $\mathcal{E}$ to the set S, as they need to get higher rank edges in subsequent stages. Note that their higher rank edges are deleted by Algorithm 1 as they become unreachable in G'_1. Thus S always stores the vertices which need to get higher rank edges in subsequent iterations.

Case 3: A rank 1 post p of a is even in G'_1: Then there is an augmenting path from a to p in H'_1. Find it and augment M_1 to get a rank-maximal matching M'_1 in H'_1. Recompute the $\mathcal{E}, \mathcal{O}, \mathcal{U}$ labels. Delete higher rank edges on those vertices whose labels change from $\mathcal{E}$ in G'_1 to $\mathcal{U}$ in H'_1.

Delete $\mathcal{OO}$ and $\mathcal{OU}$ edges if present. Add those vertices to T which are odd or unreachable in H'_1. These are precisely those vertices that will not get higher rank edges in any subsequent iteration even if they become even in one such iteration.

For each subsequent stage $i > 1$, the algorithm proceeds as follows:

---

[3] In Irving et al.'s algorithm, these labels are called $\mathcal{E}_1, \mathcal{O}_1, \mathcal{U}_1$. We omit the subscripts for the sake of bravity. The subscripts are clear from the stage under consideration.

Fig. 1. Example of status change of nodes after addition of applicant a_4. Dashed lines indicate a rank-maximal matching before addition of a_4. In (i), a_1, p_1 are unreachable before adding a_4. After adding a_4, p_1 becomes odd while a_1 becomes even. In (ii), there is no status change after adding a_4. In (iii), there is an augmenting path a_4, p_3, a_3, p_4 after adding a_4. Augmentation makes all the nodes unreachable. Preference list for each figure is shown below the figure. Note that some edges on p_1 are deleted because they are $\mathcal{OO}$ or $\mathcal{OU}$ edges.

1. Start with $H'_i = G'_i$. Add a and its undeleted edges up to rank i to H'_i.
2. If there are applicants in the set S as described in Case 2 above, add edges of rank i incident on them to H'_i.
3. Start with a matching M'_i in H'_i such that M'_i has all the edges of M'_{i-1} and those rank i edges of M_i which are not incident on any vertex matched in M'_{i-1}.
4. Check if there is an augmenting path in H'_i with respect to M'_i. If so, augment M'_i.
5. Recompute the labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$.
6. Delete higher rank edges on those vertices whose labels change from $\mathcal{E}$ to $\mathcal{U}$ or $\mathcal{O}$. Remove such vertices from S if they are present in S.
7. Delete $\mathcal{OO}$ or $\mathcal{OU}$ edges, if present. Now we have the final updated reduced graph H'_i.
8. Add those vertices from $V \setminus T$ to S whose labels change from $\mathcal{U}$ or $\mathcal{O}$ to $\mathcal{E}$. Add those vertices to T which are odd or unreachable in H'_i.

The algorithm stops when there are no more edges left in H. Figure 1 shows an example of the various cases considered above.

Analysis of the vertex-addition algorithm: Recall the notation that G is the given instance and H is the instance obtained by adding an applicant a along with its incident edges. Moreover, G_i and H_i are subgraphs of G and H respectively, consisting of edges up to rank i respectively from G and H. Also G'_i is the reduced graph corresponding to stage i of an execution of Algorithm 1 on G whereas H'_i is the graph of stage i for H constructed by the vertex-addition

algorithm. In Theorem 2, we prove that H_i' is indeed the reduced graph that would be constructed by an execution of Algorithm 1 on H.

The following Lemma is useful in analyzing the running time of the algorithm. It proves that there can be at most one new augmenting path at any stage i with respect to M_i in H_i'. Recall that M is a rank-maximal matching in G and M_i is the subset of M consisting of edges of rank only up to i. We give the proof in Appendix.

Lemma 3. *At each stage i, $|M_i| \leq |M_i'| \leq |M_i| + 1$. Thus, for any stage i, there can be at most one augmenting path with respect to M_i in H_i.*

Correctness of the algorithm is given by the following theorem, we prove its base case here, the full proof appears in Appendix.

Theorem 2. *Algorithm 2 correctly updates the rank-maximal matching and the reduced graphs. Moreover, it runs in time $O(r(m + n))$.*

Proof. We prove this by induction on ranks. Thus we prove that, if the stage-wise reduced graphs are updated correctly up to stage $i - 1$, then the algorithm correctly constructs H_i', and gives a rank-maximal matching M_i' in H_i.

As base case, consider the graph G_1' and let a be added to G_1' along with his rank 1 edges.

Case 1: Each rank 1 post p of a is odd in G_1': Then p has an alternating path from an unmatched applicant in G_1'. Addition of a only creates one more such path, so there is no augmentation and no change of labels. Applicant a remains unmatched and hence even. This can be checked in $O(1)$ time for each post using the information stored at the preprocessing stage. This case is considered in line 6 of Algorithm 2. Thus H_1' is same as G_1' with a and its rank 1 edges added. Also $M_1' = M_1$.

Case 2: A rank 1 post p of a is unreachable in G_1' but none is even: Since p is unreachable, there is no alternating path to p from an unmatched applicant or post in G_1'. Addition of the edge (a, p) creates such a path. Hence the label of p changes from $\mathcal{U}$ in G_1' to $\mathcal{O}$ in H_1'. The label on the matched partner $M(p)$ of p in M then changes from $\mathcal{U}$ to $\mathcal{E}$. There could be other applicants and posts which are unreachable in G_1' but have alternating paths from p that use the edge $(p, M(p))$. Such applicants and posts now have respectively an even and odd length alternating path from a and hence their labels change from $\mathcal{U}$ to $\mathcal{E}$ and $\mathcal{O}$ respectively. Note that these alternating paths are considered with respect to the existing matching M_1, and $M_1' = M_1$.

Consider the applicants whose labels change from $\mathcal{U}$ to $\mathcal{E}$. As these applicants are unreachable in G_1', Algorithm 1 must have deleted their higher rank edges from G. These edges need to be added back as they have become even now. We include them in the set S so that such edges can be added at the respective stages.

Case 3**: Applicant** a **has a rank** 1 **post** p **which is even in** G'_1**:** Then p has an alternating path from some unmatched post q (possibly $p = q$). This, along with a, now forms an augmenting path and M_1 needs to be augmented. This path can be found in $O(m + n)$ time by a BFS or DFS from a and M_1 is augmented to get M'_1 in the same time.

This augmentation leads to changing q from an unmatched to matched post. Now p may not have an alternating path from an unmatched post. If this happens, p becomes unreachable. Other posts on the alternating path from q to p, if any, also become unreachable and their higher rank edges need to be deleted. Their corresponding matched applicants, that were odd earlier, also become unreachable. This needs a recomputation of $\mathcal{E}, \mathcal{O}, \mathcal{U}$ labels. Also, higher rank edges on those posts whose labels change from $\mathcal{E}$ to $\mathcal{U}$ need to be deleted from H. Note that if p has an alternating path from an unmatched post in H'_1 with respect to M'_1, then there is no change of labels after augmentation of M_1.

If there are new $\mathcal{O}\mathcal{O}$ or $\mathcal{O}\mathcal{U}$ edges in H'_1, they need to be deleted. This completes the base case. The induction step is similar, and is given in Appendix.

4.2   Deletion of a vertex

Let an applicant a be deleted from the instance. The case of deletion of a post p is analogous, as explained in Section 3. Let G be the given instance and H be the updated instance. Thus $H = (\mathcal{A} \setminus \{a\} \cup \mathcal{P}, E \setminus E_a)$ where E_a is the set of edges incident on a. Let M be a rank-maximal matching in G. Also assume that the preprocessing step is executed on G and the information as mentioned in Section 3 is stored.

Description of the vertex-deletion algorithm If a is not matched in M, then M clearly remains rank-maximal in H, although the reduced graphs H'_i could differ a lot from the corresponding reduced graphs G'_i for each i. We describe the algorithm below, the pseudocode is given in Appendix.

   Initialization: $S, T = \emptyset$. These sets will be used later, as given in the following description.

Case (I): a **is matched in** M**:** Let j be the rank of the matched edge in M incident on a and Let $M(a) = p$. Thus a remains even in the execution of Algorithm 1 on G at least for j iterations. The algorithm now works as follows:

For each rank i from 1 to $j - 1$, initialize $H'_i = G'_i$ and $M'_i = M_i$. Delete edges of rank up to i incident on a from H'_i. Recompute the labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$. Delete from H the edges of rank $> i$ on those applicants whose label changes from $\mathcal{E}$ to $\mathcal{U}$. This is the final reduced graph H'_i. Add odd and unreachable vertices from H'_i to T. The set T contains those vertices that will not get higher rank edges at later stages even if their label changes to $\mathcal{E}$.

Now we come to the rank j at which a is matched in M. Initialize $H'_j = G'_j$ and $M'_j = M_j \setminus \{(a, p)\}$. Delete edges incident on a from H'_j. The following cases arise:

Case 1: p **is odd in** G'_j**:** Find an augmenting path in H'_j with respect to M'_j starting at p. Augment M'_j along this path. Recompute the labels. Delete from H the edges of rank $> j$ incident on those applicants whose labels change from $\mathcal{E}$ in G'_j to $\mathcal{U}$ in H'_j.

Case 2: p **is unreachable in** G'_j**:** Recompute the labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$. Include those posts to S whose label changes from $\mathcal{U}$ to $\mathcal{E}$. These posts need to get edges of rank $> j$ in subsequent iterations.

Case 3: p **is even in** G'_j**:** Recompute the labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$ in H'_j. Remove higher rank edges on those posts whose labels change from $\mathcal{E}$ to $\mathcal{U}$.

Add the odd and unreachable vertices from H'_j to T. Remove such vertices from S, if they are present in S. These are the vertices that will not get higher rank edges even if they get the label $\mathcal{E}$ at a later stage.

For each rank i from $j + 1$ to r, initialize $H'_i = G'_i$, except for a and its incident edges. Add edges of rank i on posts in S. Initialize $M'_i = M'_{i-1} \cup$ set of those edges in M_i which are disjoint from the edges in M'_{i-1}. Look for an augmenting path, and augment M'_i if an augmenting path is found. Recompute the labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$. Update S and T as mentiond above.

Case (II): a **is unmatched in** M**:** The algorithm involves iterating over $i = 1$ to r and computing the reduced graphs H'_i as follows: Start with $H'_i = G'_i$, deleting a and its incident edges from H'_i, add rank i edges on vertices in S, recompute the labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$, include those vertices from $V \setminus T$ into set S whose labels change from $\mathcal{U}$ to $\mathcal{E}$. Add vertices with $\mathcal{O}$ or $\mathcal{U}$ labels to T. Delete higher rank edges on the vertices whose labels are $\mathcal{O}$ or $\mathcal{U}$.

The correctness of the vertex-deletion algorithm is given by the theorem below. The proof and an example (Figure 2 appear in Appendix.

Theorem 3. *Algorithm 3 correctly updates the rank-maximal matching M on deletion of an applicant. Moreover, it takes time $O(r(m + n))$.*

4.3   Addition and deletion of an edge

Modules similar to those for vertex-addition and vertex-deletion can be written for addition and deletion of an edge, which would have time complexity $O(r(m + n))$ each. However, both edge-addition and edge -deletion can be performed as a vertex-deletion followed by vertex-addition, achieving the same running time $O(r(m + n))$. We explain this here. To add an edge (a, p), one can first delete applicant a using the vertex-deletion algorithm thereby deleting all the edges E_a incident on a, and then the applicant a is added back along with the edge-set $E_a \cup \{(a, p)\}$. Similarly, deletion of an edge (a, p) can be carried out by first deleting the applicant a along with the set of edges E_a incident on a and then adding back a along with the edge-set $E_a \setminus \{(a, p)\}$. It is clear that both edge-addition and edge-deletion can thus be carried out in $O(r(m + n))$ time.

5   Discussion

In this paper, we give an $O(r(m + n))$ algorithm to update a rank-maximal matching when vertices or edges are added and deleted over time. Independent

of our work, [8] give an algorithm for vertex addition and deletion that runs in $O(m)$ time using similar techniques.

In [9], a switching graph characterization of rank-maximal matchings has been developed, which has found several applications. A natural question to ask is whether this characterization is also useful in dynamic setting. However, a switching graph is based on the *reduced graph* computed by Irving et al.'s algorithm, which is a subgraph of the input graph. Addition or deletion of a vertex can change this subgraph and hence the switching graph significantly. Therefore it is not immediate whether the switching graph characterization can help in dynamic setting. It is an interesting question to explore.

References

1. D. J. Abraham, K. Cechlárová, D. F. Manlove, and K. Mehlhorn. Pareto-optimality in house allocation problems. In *Proceedings of 15th ISAAC*, pages 3–15, 2004.
2. D. J. Abraham, R. W. Irving, T. Kavitha, and K. Mehlhorn. Popular matchings. *SIAM Journal on Computing*, 37(4):1030–1045, 2007.
3. D. J. Abraham and T. Kavitha. *Dynamic Matching Markets and Voting Paths*, pages 65–76. 2006.
4. S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in o(log n) update time. *SIAM J. Comput.*, 44(1):88–113, 2015.
5. S. Bhattacharya, M. Henzinger, and G. F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 785–804, 2015.
6. S. Bhattacharya, M. Henzinger, and D. Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 398–411, 2016.
7. S. Bhattacharya, M. Hoefer, C.-C. Huang, T. Kavitha, and L. Wagner. Maintaining near-popular matchings. In *Automata, Languages, and Programming: 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 504–515, 2015.
8. P. Ghosal, A. Kunysz, and K. Paluch. The dynamics of rank-maximal and popular matchings. *CoRR*, abs/1703.10594, 2017.
9. P. Ghosal, M. Nasre, and P. Nimbhorkar. *Rank-Maximal Matchings – Structure and Algorithms*, pages 593–605. 2014.
10. M. Gupta and R. Peng. Fully dynamic (1+ e)-approximate matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013*, pages 548–557, 2013.
11. A. Hylland and R. Zeckhauser. The efficient allocation of individuals to positions. *Journal of Political Economy*, 87(2):293–314, 1979.
12. R. W. Irving. Greedy matchings. *Technical Report, University of Glasgow*, TR-2003-136, 2003.
13. R. W. Irving, T. Kavitha, K. Mehlhorn, D. Michail, and K. E. Paluch. Rank-maximal matchings. *ACM Transactions on Algorithms*, 2(4):602–610, 2006.

14. T. Kavitha and C. D. Shah. Efficient algorithms for weighted rank-maximal matchings and related problems. In *Proceedings of 17th ISAAC*, pages 153–162, 2006.

15. K. Onak and R. Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 457–464, 2010.

16. K. E. Paluch. Capacitated rank-maximal matchings. In *Proceedings of 8th CIAC*, pages 324–335, 2013.

17. W. R. Pulleyblank. Handbook of combinatorics (vol. 1). chapter Matchings and Extensions, pages 179–232. MIT Press, Cambridge, MA, USA, 1995.

18. Y. Yuan. Residence exchange wanted: A stable residence exchange problem. *European Journal of Operational Research*, 90(3):536 – 546, 1996.

A Example for addition of an edge

We give an example to show that addition of an edge can change the rank-maximal matching by $\Omega(n)$ edges.

Let the given instance be as follows:

$$a_1 : p_1$$
$$a_2 : p_5, p_1, p_2$$
$$a_3 : p_5, p_6, p1, p_2, p_3$$
$$a_4 : p_5, p_6, p_1, p_7, p_2, p_3, p_4$$
$$a_5 : p_5$$
$$a_6 : p_6, p_8$$
$$a_7 : p_7$$

The instance has only one rank-maximal matching given by $M = \{(a_1, p_1), (a_2, p_2), (a_3, p_3), (a_4, p_4), (a_5, p_5), (a_6, p_6), (a_7, p_7)\}$

Now consider addition of an edge (a_1, p_8) of rank 1, so that the instance becomes

$$a_1 : (p_1, p_8)$$
$$a_2 : p_5, p_1, p_2$$
$$a_3 : p_5, p_6, p_1, p_2, p_3$$
$$a_4 : p_5, p_6, p_1, p_7, p_2, p_3, p_4$$
$$a_5 : p_5$$
$$a_6 : p_6, p_8$$
$$a_7 : p_7$$

This new instance also admits only one rank-maximal matching M' given by $M' = \{(a_1, p_8), (a_2, p_1), (a_3, p_2), (a_4, p_3), (a_5, p_5), (a_6, p_6), (a_7, p_7)\}$

Note that M and M' differ by 4 edges, which is more than half the size of M or M'. The example can be easily scaled for any number of applicants.

B Details of vertex-addition

Lemma 3 *At each stage i, $|M_i| \leq |M_i'| \leq |M_i|+1$. Thus, for any stage i, there can be at most one augmenting path with respect to M_i in H_i.*

Proof. Recall from invariant $(I3)$ of Algorithm 1 mentioned in Section 2 that M_i and M_i' are the rank-maximal matchings in G_i and H_i respectively. Here G_i and H_i are the instances G and H with only the edges of ranks 1 to i present.

Consider $M_i \oplus M_i'$, which is the set of edges present in exactly one of the two matchings. This is a collection of vertex-disjoint paths and cycles. Each

---
Algorithm 2 Update algorithm for addition of a new applicant a
---

1: $S = \emptyset, T = \emptyset$

2: **for** each rank i from 1 until a is matched **do**

3:     Update G'_i to get H'_i: If there are vertices in S (added in step 9 of previous iteration), add rank i edges incident on them. These are the applicants which changed from $\mathcal{U}$ to $\mathcal{E}$ in one of the previous stages. Update the $\mathcal{O}, \mathcal{U}, \mathcal{E}$ labels.

4:     Add edges between a and those of his rank i posts which do not become odd or unreachable in H'_j for any $j < i$.

5:     **if** All of a's rank i posts are odd in G'_i **then**

6:        There is no change in labels, a remains even. Do nothing.

7:     **else if** One or more of a's rank i posts are unreachable in G'_i and no rank i post of a is even in G'_i **then**

8:        Recompute the labels.

9:        Now some unreachable posts become odd and corresponding unreachable applicants become even. Include these applicants to S for addition of higher rank edges later if they are not present in T.

10:     **else if** One of a's rank i posts is even in G'_i **then**

11:        Augment M_i by finding an augmenting path from a. Call this matching M'_i.

12:        Recompute the labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$. {/* Now some even posts may become unreachable. Some odd applicants may become unreachable. The applicant a will be odd or unreachable.*/}

13:        Delete higher rank edges on the posts whose labels change from $\mathcal{E}$ to $\mathcal{U}$.

14:        Delete $\mathcal{OU}$ and $\mathcal{OO}$ edges from H'_i.

15:        Remove those vertices from S become $\mathcal{O}$ or $\mathcal{U}$ now. This is the updated graph H'_i.

16:        Add odd or unreachable vertices in H'_i to T.

17:     **end if**

18: **end for**

19: Now a is matched to some post p by a rank i edge in M'_i.

20: **for** each rank $j = i + 1$ to r **do**

21:     Start with $H'_j = G'_j$. Add edges of rank j incident on vertices in S to H'_j.

22:     Update M_j to reflect the changes that were made at earlier stages, and call it M'_j.

23:     Search for an augmenting path in H'_j with respect to M'_j. Augment M'_j if an augmenting path is found.

24:     Relabel vertices and delete $\mathcal{OO}$ and $\mathcal{OU}$ edges from H'_j. {/*This is the final H'_j. Also M'_j is the rank-maximal matching in H_j.*/}

25:     Remove the vertices from S which are now odd or unreachable.

26:     Add odd or unreachable vertices to T. Delete higher rank edges on them from H.

27: **end for**

---

path that does not contain the new applicant a, and each cycle must have the same number of edges of each rank from M_i and M'_i. Otherwise we can obtain a matching which has a better signature than either M_i or M'_i, which contradicts the rank-maximality of both the matchings in G_i and H_i respectively. At most one path can have the new applicant a as one end-point. This path can contain at most one more edge of M'_i than that of M_i. This proves the first part.

To see that there can be an augmenting path at multiple stages, consider the case where a post p gets matched to the new applicant a at stage i. If p is matched to an applicant b in M and the edge (b, p) has rank j such that $j > i$, then b is matched in G'_j but not in H'_j. Hence there can possibly be a new augmenting path in H'_j starting at b. □

Theorem 2 *Algorithm 2 correctly updates the rank-maximal matching and the reduced graphs. Moreover, it runs in time $O(r(m + n))$.*

Proof. We prove this by induction on ranks. Thus we prove that, if the stage-wise reduced graphs are updated correctly up to stage $i - 1$, then the algorithm correctly constructs H'_i, and gives a rank-maximal matching M'_i in H_i.

As base case, consider the graph G'_1 and let a be added to G'_1 along with his rank 1 edges.

Case 1: Each rank 1 post p of a is odd in G'_1: Then p has an alternating path from an unmatched applicant in G'_1. Addition of a only creates one more such path, so there is no augmentation and no change of labels. Applicant a remains unmatched and hence even. This can be checked in $O(1)$ time for each post using the information stored at the preprocessing stage. This case is considered in line 6 of Algorithm 2. Thus H'_1 is same as G'_1 with a and its rank 1 edges added. Also $M'_1 = M_1$.

Case 2: A rank 1 post p of a is unreachable in G'_1 but none is even: Since p is unreachable, there is no alternating path to p from an unmatched applicant or post in G'_1. Addition of the edge (a, p) creates such a path. Hence the label of p changes from $\mathcal{U}$ in G'_1 to $\mathcal{O}$ in H'_1. The label on the matched partner $M(p)$ of p in M then changes from $\mathcal{U}$ to $\mathcal{E}$. There could be other applicants and posts which are unreachable in G'_1 but have alternating paths from p that use the edge $(p, M(p))$. Such applicants and posts now have respectively an even and odd length alternating path from a and hence their labels change from $\mathcal{U}$ to $\mathcal{E}$ and $\mathcal{O}$ respectively. Note that these alternating paths are considered with respect to the existing matching M_1, and $M'_1 = M_1$.

Consider the applicants whose labels change from $\mathcal{U}$ to $\mathcal{E}$. As these applicants are unreachable in G'_1, Algorithm 1 must have deleted their higher rank edges from G. These edges need to be added back as they have become even now. We include them in the set S so that such edges can be added at the respective stages. This is done in lines 8 and 9 of Algorithm 2 and edges on applicants in S are added in line 3.

Case 3: Applicant a has a rank 1 post p which is even in G'_1: Then p has an alternating path from some unmatched post q (possibly $p = q$). This, along with a, now forms an augmenting path and M_1 needs to be augmented. This path can be found in $O(m + n)$ time by a BFS or DFS from a and M_1 is augmented to get M'_1 in the same time.

This augmentation leads to changing q from an unmatched to matched post. Now p may not have an alternating path from an unmatched post. If this

happens, p becomes unreachable. Other posts on the alternating path from q to p, if any, also become unreachable and their higher rank edges need to be deleted. Their corresponding matched applicants, that were odd earlier, also become unreachable. This needs a recomputation of $\mathcal{E}, \mathcal{O}, \mathcal{U}$ labels. Also, higher rank edges on those posts whose labels change from $\mathcal{E}$ to $\mathcal{U}$ need to be deleted from H. This is dealt with in lines 11 to 15 of Algorithm 2. Note that if p has an alternating path from an unmatched post in H_1' with respect to M_1', then there is no change of labels after augmentation of M_1.

If there are new $\mathcal{OO}$ or $\mathcal{OU}$ edges in H_1', they need to be deleted.

The algorithm also stores those vertices which are odd or unreachable in H_1' in a set T. These are precisely those vertices that will not be added to S at any point and hence will not get higher rank edges later. Thus at the end of stage 1, we have the graph H_1' exactly same as what would be given by executing Algorithm 1 on H. We also have a maximum matching M_1' in H_1, which is trivially rank-maximal when edges up to rank 1 are considered.

We now come to the inductive part. Assume that the algorithm has correctly computed H_j' for $1 \leq j < i$. We show that the algorithm then correctly computes H_i' and M_i'.

Initialization The algorithm starts from $H_i' = G_i'$ and the matching M_i' in H_i is initialized to $M_{i-1}' \cup$ set of those edges in M_i that are vertex-disjoint from edges in M_{i-1}'. Note that there could be a vertex that is matched in M_{i-1}' but not in M_{i-1}, and possibly matched in M_i. Thus the initial matching M_i' is same as M_i except for the updates performed in earlier stages.
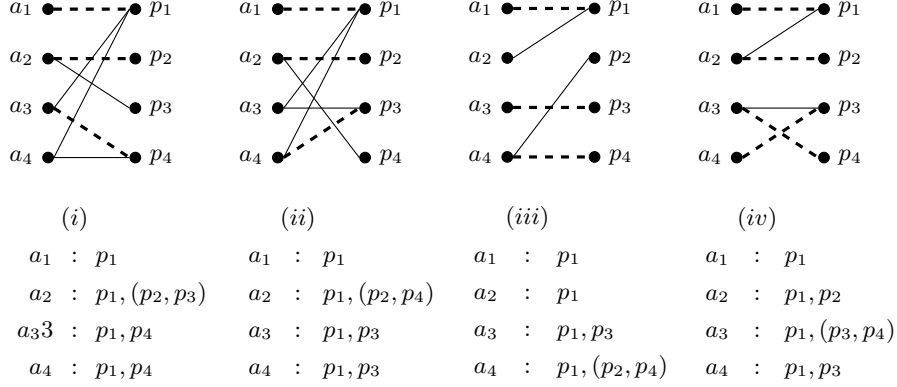
Recall that S is the set of vertices which do not have rank i edges in G_i' but need to get rank i edges in H_i'. The algorithm adds rank i edges on such vertices. It also adds applicant a and its undeleted edges to H_i'.

Checking for augmenting path: If a is still unmatched, then there could be an augmenting path in H_i' starting at a. Even if a is matched in M_{i-1}', there could still be an augmenting path in H_i' with respect to M_i', as explained below:

If a is matched in M_{i-1}', say by a rank $j \leq i-1$ edge, then there is also a post that is matched in M_{i-1}' but not in M_{i-1}, say q. This is because augmentation along an augmenting path always matches an additional applicant (in this case, a) and an additional post (in this case q). However, in M, i.e. prior to addition of a, q may have been matched to some applicant b at a rank $k > j$. Now q is matched to a, so b loses its matched edge at stage k. This needs updating labels $\mathcal{O}, \mathcal{U}, \mathcal{E}$ at subsequent stages. Also, we need to find an augmenting path from b, if any, at a later stage. This is done in lines 21 to 25.

Note that there can be at most one augmenting path according to Lemma 3.

Recomputation of labels: Thus, at each stage, the algorithm looks for an augmenting path and augments M_i', if such a path is found. The augmentation can lead to change of labels, and deletion of edges on those vertices

	(i)			(ii)			(iii)			(iv)	
a_1	:	p_1	a_1	:	p_1	a_1	:	p_1	a_1	:	p_1
a_2	:	$p_1, (p_2, p_3)$	a_2	:	$p_1, (p_2, p_4)$	a_2	:	p_1	a_2	:	p_1, p_2
a_33	:	p_1, p_4	a_3	:	p_1, p_3	a_3	:	p_1, p_3	a_3	:	$p_1, (p_3, p_4)$
a_4	:	p_1, p_4	a_4	:	p_1, p_3	a_4	:	$p_1, (p_2, p_4)$	a_4	:	p_1, p_3

Fig. 2. Figure indicating possible status changes after deletion of applicant a_4: Applicants are shown on left whereas posts are shown on right. Dashed lines indicate a rank-maximal matching M prior to deletion of a_4. Preference lists are shown below each figure. In (i), a_4 is unmatched. Deletion of a_4 keeps M unchanged but status of a_3 and p_4 changes respectively from even and odd to unreachable. Edge (a_3, p_1) needs to be deleted. In (ii), a_4 is matched and even. Deletion of a_4 results in augmenting path and M changes to $(M \setminus \{(a_4, p_3)\}) \cup \{(a_3, p_3)\}$. In (iii), a_4 is odd. Deletion of a_4 does not change the status of any node. In (iv), a_4 is unreachable. Deletion of a_4 changes the status of p_3, p_4 from unreachable to even and that of a_3 from unreachable to odd. Note that some edges incident on p_1 have been deleted as they are $\mathcal{OO}$ or $\mathcal{OU}$ edges.

whose labels change from $\mathcal{E}$ to $\mathcal{U}$ due to the augmentation. Also, even if there is no augmentation, there could still be a change of labels due to addition of edges on vertices in S and also due to addition of edges incident on a. Thus the labels need to be recomputed anyway. The sets S and T are updated as mentioned in the base case above.

As all the possible differences between G'_i to H'_i are considered above, H'_i is the correct reduced graph of stage i. Further, by Lemma 3, M'_i is a maximum matching in H_i obtained by augmenting a rank-maximal matching M'_{i-1} from H'_{i-1}. Thus M'_i is rank-maximal in H_i by correctness of Algorithm 1.

Each of the three operations described above need $O(m+n)$ time. Whenever the label on a vertex changes, or an edge is deleted, the stored information can be updated in $O(1)$ time. Thus each stage can be updated in time $O(m+n)$, so total update time is $O(r(m+n))$. $\qquad\square$

C Details of vertex-deletion

Theorem 3 *Algorithm 3 correctly updates the rank-maximal matching M on deletion of an applicant. Moreover, it takes time $O(r(m+n))$.*

Proof. If a is unmatched in M, clearly M remains unchanged by deletion of a. As a is assumed to be matched to a post p by a rank j edge, a is even at least for the first $j - 1$ stages of Algorithm 1. Hence the applicants and posts which have alternating paths from a are respectively even and odd in all those stages. Deletion of a may make them unreachable, if they do not have alternating paths from another unmatched applicant. This needs recomputation of labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$. Also, higher rank edges on those applicants whose label changes from $\mathcal{E}$ to $\mathcal{U}$ need to be deleted. This is done in steps 4 to 6 of Algorithm 3.

At stage j, deletion of a leads to deletion of the edge (a, p) from M. Hence p becomes free. The following cases arise:

Case 1: p is odd in G'_j: Then there is an alternating path to p from some unmatched applicant in G'_j with respect to M_j. This path now becomes an augmenting path in H'_j with respect to M'_j. Checking this case and augmenting along an augmenting path starting from p takes $O(m + n)$ time. Now the posts on this path may not have an alternating path from an unmatched applicant. In this case, their labels change from $\mathcal{O}$ to $\mathcal{U}$ and those of the applicants matched to them change from $\mathcal{E}$ to $\mathcal{U}$. Thus higher rank edges on such applicants need to be deleted. This is done in lines 11 to 13.

Case 2: p is unreachable in G'_j: Then there is no alternating path to p with respect to M_j from an unmatched applicant and hence no augmentation is possible at this stage. In this case, p remains unmatched in H'_j, and hence has the label $\mathcal{E}$. Labels on applicants and posts which have alternating paths from p change from $\mathcal{U}$ to $\mathcal{O}$ and $\mathcal{U}$ to $\mathcal{E}$ respectively. Such posts need to get back their higher rank edges which were deleted in Algorithm 1. We include such posts into the set S. This takes $O(m + n)$ time and is done in lines 12 to 15.

Case 3: p is even in G'_j: Thus p and possibly some more vertices have an alternating path with respect to M_j in G'_j from an unmatched post q. Due to deletion of (a, p) edge, some of these vertices may no longer have an alternating path from q. Labels on such applicants and posts change from $\mathcal{O}$ and $\mathcal{E}$ respectively to $\mathcal{U}$. The algorithm deletes higher rank edges on such posts.

Now the algorithm considers subsequent stages. If p is odd in G'_j above and the matching is augmented as described above, it leads to matching an applicant b in M'_j who is unmatched in M_j. If, in M, b is matched to say q at a rank $k > j$ then the augmentation will lead to q losing its matched edge at stage k. In this case, same procedure needs to be repeated as above.

Thus the algorithm runs in time $O(m + n)$ for each stage and hence a total of $O(r(m + n))$ time. As all cases are exhaustively considered above, it updates the matching and the reduced graphs correctly. □

Algorithm 3 Update algorithm for deletion of an applicant a

1: Let p be the post to which a is matched by a rank j edge in M.
2: $S = T = \emptyset$
3: **for** each rank $i = 1$ to $j - 1$ **do**
4:     Initialize $H_i' = G_i'$, $M_i' = M_i$. Remove a and its incident edges from H_i'. {/*Certainly a is even in G_i'./*}
5:     Recompute the labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$.
6:     This can change some posts from $\mathcal{O}$ to $\mathcal{U}$ and their matched applicants from $\mathcal{E}$ to $\mathcal{U}$. Delete higher rank edges on such applicants from H.
7:     Include vertices with labels $\mathcal{O}$ and $\mathcal{U}$ into T.
8: **end for**
9: Initialize $H_j' = G_j'$, $M_j' = M_j \setminus \{(a, p)\}$. Remove a and its incident edges from H_j'. {This makes post p unmatched and hence even.}
10: **if** p is odd in G_j' **then**
11:     Find an augmenting path from p in H_j' respect to M_j' and augment M_j'.
12:     Recompute the labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$.
13:     Labels on some applicants may change from $\mathcal{E}$ to $\mathcal{U}$. Delete higher rank edges on them. Labels on some posts may change from $\mathcal{O}$ to $\mathcal{U}$.
14: **else if** p is unreachable in G_j' **then**
15:     Recompute the labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$ in H_j' with respect to M_j'.
16:     Labels on some posts including p change from $\mathcal{U}$ to $\mathcal{E}$. Include these posts into S for addition of higher rank edges in later stages unless they are in T.
17:     Labels on the applicants matched to these posts change from $\mathcal{U}$ to $\mathcal{O}$.
18: **else if** p is even in G_j' **then**
19:     Recompute the labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$ in H_j' with respect to M_j'.
20: **end if**
21: **for** each rank $i = j + 1$ to r **do**
22:     Initialize $H_i' = G_i'$,
23:     $M_i' = M_{i-1}' \cup$ set of those edges in M_i which are disjoint from edges in M_{i-1}'.
24:     Remove a and its incident edges from H_i' and M_i'. Add rank i edges on posts in S, if any.
25:     Check for an augmenting path in H_i' with respect to M_i' and augment M_i' if such an augmenting path is found.
26:     Recompute the labels $\mathcal{E}, \mathcal{O}, \mathcal{U}$ in H_i' with respect to M_i'.
27:     Include
28:
29: **end for**