# Online bipartite matching in offline time

Bartłomiej Bosek*, Dariusz Leniowski†, Piotr Sankowski† and Anna Zych†
*Theoretical Computer Science Department, Faculty of Mathematics and Computer Science,
Jagiellonian University, Kraków, Poland, Email: bosek@tcs.uj.edu.pl
†Institute of Computer Science, University of Warsaw, Warszawa, Poland, Email: {d.leniowski,sank,anka}@mimuw.edu.pl

***Abstract*—This paper investigates the problem of maintaining maximum size matchings in incremental bipartite graphs. In this problem a bipartite graph $G$ between $n$ clients and $n$ servers is revealed online. The clients arrive in an arbitrary order and request to be matched to a subset of servers. In our model we allow the clients to switch between servers and want to maximize the matching size between them, i.e., after a client arrives we find an augmenting path from a client to a free server. Our goals in this model are twofold. First, we want to minimize the number of times clients are reallocated between the servers. Second, we want to give fast algorithms that recompute such reallocation.**

**As for the number of changes, we propose a greedy algorithm that chooses an augmenting path $\pi$ that minimizes the maximum number of times each server in $\pi$ was used by augmenting paths so far. We show that in this algorithm each server has its client reassigned $O(\sqrt{n})$ times. This gives an $O(n^{3/2})$ bound on the total number of changes, what gives a progres towards the main open question risen by Chaudhuri *et al.* (INFOCOM'09) who asked to prove $O(n \log n)$ upper bound. Next, we argue that the same bound holds in the decremental case. Moreover, we show incremental and decremental algorithms that maintain $(1-\varepsilon)$-approximate matching with total of $O(\varepsilon^{-1}n)$ reallocations, for any $\varepsilon > 0$.**

**Finally, we address the question of how to efficiently compute paths given by this greedy algorithm. We show that by introducing proper amortization we can obtain an incremental algorithm that maintains the maximum size matching in total $O(\sqrt{n}m)$ time. This matches the running time of one of the fastest static maximum matching algorithms that was given by Hopcroft and Karp (SIAM J. Comput '73). We extend our result to decremental case where we give the same total bound on the running time. Additionally, we show $O(\varepsilon^{-1}m)$ time incremental and decremental algorithms that maintain $(1-\varepsilon)$-approximate matching for any $\varepsilon > 0$. Observe that this bound matches the running time of the fastest approximate static solution as well.**

***Index Terms*—online matchings; bipartite matchings; incremental and decremental algorithms; approximate matchings;**

## I. Introduction

In this paper, we study an incremental bipartite matching problem, which corresponds to a scenario in which clients arrive online and request service from a set of given servers. Each client arrives together with a set of servers being able to handle his request. In this scenario we want to provide the service to as many clients as possible. Hence, when necessary we reallocate the clients between the servers, and the cost of the solution is the total number of times a client is allocated to a server during the course of this reallocation process. The solution to this problem needs to find an augmenting path (a path that alternates between matched and unmatched edges) in the current graph from the client to some unused server. Using this language we can define, in an equivalent way, the cost to be equal to the total length of all augmenting paths. This approach contrasts with the model introduced by Karp, Vazirani and Vazirani in [14], where clients are assigned to servers permanently. Our model was introduced by Grove *et al.* [8] in '95, where the authors applied competitive analysis to show an $O(\log n)$ bound when each client was connecting to at most two servers. Next, the model was studied by Chaudhuri *et al.* [7] where a total bound of $O(n \log n)$ was shown in some very restricted models, e.g., forests or random graphs with degree $\Theta(\log n)$. These special cases led the authors to conjecture that a greedy algorithm that chooses the shortest augmenting path achieves a bound of $O(n \log n)$. However, they were unable to make any progress in the general case where only the trivial bound of $O(n^2)$ is known. Hence, our result constitutes a step towards resolving this main open problem from [7]. Here, we propose a new greedy algorithm for this problem. It uses the notion of a rank of a server-vertex, which denotes the number of times the server was used by an augmenting path so far. In each step it chooses an augmenting path $\pi$ that minimizes the maximum rank of vertices on $\pi$. We prove that this algorithm reallocates each client at most $O(n^{1/2})$ times which gives the total bound of $O(n^{3/2})$.

As argued in [7] the above model has a wide set of applications in very diverse areas of computer science, e.g., streaming content delivery, web hosting, remote data storage, job scheduling, hashing, etc. In all these applications we have some clients (jobs for scheduling, or objects for hashing) arriving online that have to be allocated to some fixed set of servers (table cells for hashing). In all these cases we do not want to drop clients from service, but prefer to reallocate them so that as many clients as possible are satisfied. Here, the cost of not serving a client is much higher then the cost of reallocating the client to another server (e.g., copying an object in a hash table is cheap). Still we would like to minimize the cost of these changes. It is somewhat astonishing that despite the usefulness of this model no better solution then the naive $O(n^2)$ one was known till now.

Besides solving the incremental case, we extend our results in a twofold way. First, we show that the same bound holds in the decremental case, what follows by an easy reduction. Second, we show an incremental as well as decremental algorithm that is able to maintain $(1-\varepsilon)$-approximate matching using $O(\varepsilon^{-1}n)$ changes, for any $\varepsilon > 0$. This shows that by

This work was partially supported by ERC StG project PAAl 259515, FET IP project MULTIPEX 317532, NCN grant 2013/11/D/ST6/03100 and N206 567940.

dropping an arbitrary small, but constant fraction of clients, an optimal linear bound on the number of changes can be obtained. Previously, in this incremental approximate model the best solution was either to use the algorithm of [14] to get $(1-1/e)$-approximation in $O(n)$ reallocations, or to maintain exact matching in $O(n^2)$ reallocations.

So far we have been only counting the number of changes to the matching. Let us now change the perspective a bit and let us now consider the time needed to find these augmenting paths. Observe that our greedy algorithm can be implemented using bottleneck shortest paths, which can be found in $O(m + n \log n)$ time using Dijkstra's algorithm. Hence, a naive implementation of our algorithm would be very inefficient as it would require $O(nm + n^2 \log n)$.[1] However, our algorithm has an important property that the greedy paths it finds can be defined locally. In other words, we can show that no global information (e.g., shortest path distances) needs to be maintained in the graph to find the greedy paths. This allows us to devise a path search procedure that amortizes the required work in an efficient way. Intuitively the idea is to assign to each fixed vertex a number called *rank* that will only increase. Each time we try to look for a path that minimizes the ranks, and if we do not find one we raise them by one. We know that there always exists an augmenting path of rank bounded by $O(\sqrt{n})$, so either we find a path or exceed this bound and know that there is no augmenting path. This leads to incremental and decremental algorithms that maintain maximum matching in total $O(\sqrt{n}m)$ time. Previously, only the trivial $O(nm)$ time solution was known. With this respect it is somewhat surprising that our algorithm finds augmenting paths one by one without looking on the whole graph when doing so. In particular, our running time matches one of the best offline solutions to the problem [10], but within the same bound computes not only one maximum matching, but $n$ of them. Our algorithm can be extended to maintain $(1-\varepsilon)$-approximate matching in $O(\varepsilon^{-1}m)$ total time, for any $\varepsilon > 0$. Again this solution improves over the $O(m)$ time $(1-1/e)$-approximate algorithm implied by [14] and matches the running time of the best offline solution.[2]

Finally we extend the algorithm to graphs with integer edge weights bounded by $W$, where it is able to maintain maximum weight matchings in total $O(W^{3/2}\sqrt{n}m)$ time, or $(1-\varepsilon)$-approximation of them in total $O(W\varepsilon^{-1}m)$ time.

The bipartite matching problem is a central problem in combinatoric optimization and has a wide variety of applications, as it models the paradigmatic case of allocating objects to agents. Hence, having a tool to maintain such allocations in an incremental case is a very useful instrument. With this respect we note that our algorithms are purely combinatorial and very easy to implement. Implementation wise, our solution in some aspects is easier then the Hopcroft-Karp algorithm. We use only one graph search procedure (e.g., DFS or BFS), whereas Hopcroft-Karp algorithm uses alternative calls to both DFS and BFS. Efficiency wise, our algorithms should perform in practice in a similar way as Hopcroft-Karp algorithm. We are not going to review all possible applications here, let us just mention one that gained a lot of attention recently, i.e., the AdWords problem [15]. In this application we indeed need to take care of vertices that arrive online. Moreover, we note that the algorithms that allocate AdWords might be guided by a solution constructed incrementally from previous requests. However, due to the large data size it is impossible to recompute each time the optimal solution, and only $(1-1/e)$-approximate allocation can be used [16]. Hence, our framework could deliver tools that would allow to maintain better, or even optimal, solution in this application.

*a) Related Work:* Our work is most closely related to two lines of research. The first one, as discussed above, is the study of online matchings with augmentations [8], [7]. The second one is the recent series of papers on dynamic matchings [12], [20], [18], [4], [17], [9], [2]. One should note, however, that in all these papers a different dynamic model is studied. First, these papers consider edge updates (insertion and deletion of edges), whereas we consider vertex updates. As pointed above this model is closer to some applications then edge updates scenario, this was pointed as well in [5], [6], where it was noted that vertex-update dynamic problems are often more challenging. Second, all of them consider a more general fully-dynamic model, where the sequence of updates consists of intermixed insert and delete operations. When comparing these results with our incremental or decremental ones we derive the total time needed to handle $m$ edge updates using these approaches. We first note that the solutions given in [12], [20] lose against the $O(nm)$ time algorithms, that is obtained by executing a search for an augmenting path each time a vertex is added or removed. On the other hand, the solution of [18] is superseded by [4] both in the running time and approximation factor. The solution of [4] results in a decremental as well incremental $1/2$-approximate algorithm working in total of $O(m \log n)$ time. Here, we improve this by giving $(1-\varepsilon)$-approximation in $O(\varepsilon^{-1}m)$ total time. As for both [17], [9], the algorithm in [9] gives a better approximation than [17] in essentially the same running time. When translated into our model these result imply an algorithm that maintains $(1-\varepsilon)$-approximate matching in $O(\varepsilon^{-2}m\sqrt{m})$ total time. Our results compare well by giving either the same guarantee in much smaller $O(\varepsilon^{-1}m)$ total time, or by allowing to maintain exact maximum in better $O(\sqrt{n}m)$ time. As for the weighted case none of the papers [17], [9], [2], where the problem is studied, is able to dynamically maintain exact maximum weight matching even when weights are bounded by a constant. On the other hand, we note that [9] implies an $O(W\varepsilon^{-2}\sqrt{m}m)$ time algorithm that maintains $(1-\varepsilon)$-approximate matching. Here, again our $O(W\varepsilon^{-1}m)$ time algorithm gives a visible improvement.

Recently there was some progress on showing lower bound on the running time of dynamic algorithms [19], [1]. The second of these papers basing on some conjectures shows

[1]The same bound holds for the algorithm proposed in [7], but there seem to be no possibility to improve it.

[2]It is folklore that running $O(\varepsilon^{-1})$-phases of the Hopcroft-Karp algorithm results in $(1-\varepsilon)$-approximate matching and takes $O(\varepsilon^{-1}m)$ time.

polynomial lower bounds on the running time of incremental and decremental maximum matching algorithms. However, these result do not apply to our model, as they work with edge updates. There is no known reduction going in the required direction, because intuitively it would require to add vertices on both sides of the graph.

Finally, we note that our model is related to the online load balancing of permanent task with preemption. This model has been studied in many papers (see [3] for a survey). However, in our model we assume hard capacity constraints, whereas in the load balancing models one usually assumes that the capacities are soft, i.e., the capacities can be exceeded and one is interested in minimizing the maximum load. Hence, these results have somewhat different objectives.

*b) Paper Overview:* In the next section we formally define our problem, and introduce the basic framework that will be employed by our greedy approach. In Section III we prove some basic properties of ranks in this framework, whereas in Section IV basing on these properties we give the $O(\sqrt{n})$ upper bound on maximum ranks. Next, in Section V we give all the details of the amortized implementation of our algorithms. In Section VI we describe extensions of our result to approximate, as well as, to decremental and weighted scenario. Finally, Section VII concludes the paper and gives some interesting open problems.

## II. Preliminaries

In this section we formally define the online maximum matching problem in bipartite graphs and introduce the general framework we use to approach this problem. Let $W$ and $B$ be two sets of vertices over which the bipartite graph will be formed. The set $W$ is given up front to the algorithm, i.e., corresponds to servers, whereas the vertices in $B$ (clients) arrive online. Throughout the paper, the vertices of $W$ are referred to as white and the vertices of $B$ are referred to as black. We denote by $G_t = \langle W \uplus B_t, E_t \rangle$ the bipartite graph after the $t$'th black vertex has arrived. The graph $G_t$ is constructed online in the following manner:

- we start with $G_0 = \langle W \uplus B_0, E_0 \rangle = \langle W \uplus \varnothing, \varnothing \rangle$;
- in turn $t$ a new vertex $b_t \in B$ together with all its incident edges $E(b_t)$ is revealed and $G_t$ is defined as:

$$\begin{cases} E_t = E_{t-1} \cup E(b_t), \\ B_t = B_{t-1} \cup \{b_t\}\,; \end{cases}$$

- we finish when $t = |W|$, i.e., the number of black vertices is equal to the number of white vertices.

The goal of our algorithm is to compute for each $G_t$ the maximum size matching $M_t$. The final graph $G_{|W|}$ which is obtained in this process will be denoted by $G = (W \uplus B, E)$. We denote $n = |W| = |B|$ and $m = |E|$.

For every $t \in [n]$, we add orientation to edges of the graph $G_t$. This orientation is induced by matching $M_t$: the matched edges are oriented towards black vertices, while the unmatched edges are oriented towards white vertices. When a new vertex $b_t$ arrives, we get an intermediate orientation $G_t^{\text{int}} = (E_t^{\text{int}}, B_t)$, where the edges of $b_t$ are oriented towards its neighbors, and the rest of the edges is oriented according to $M_{t-1}$. Note that $G_t^{\text{int}}$ and $G_{t-1}$ differ only by one vertex $b_t$. Any simple directed path in $G_t^{\text{int}}$ from $b_t$ to some unmatched white vertex is an augmenting path. In turn $t$, if $b_t$ can be matched, the edges of $G_t^{\text{int}}$ are reoriented along some augmenting path chosen by the algorithm, and the resulting orientation is $G_t$. The unmatched white vertices are called *seeds*. We denote the set of seeds in turn $t$ as

$$S_t = \{w \in W : wb \notin M_t \text{ for any } b \in B\}\,.$$

So in turn $t$ the augmenting paths in $G_t^{\text{int}}$ are the directed paths from $b_t$ to some $s \in S_{t-1}$. We represent a path as an ordered sequence of vertices. We use the notation $v \xrightarrow{\pi} v'$ to denote that a (directed) path $\pi$ starts in $v$ and ends in $v'$. We use the notation $v \in \pi$ and $\rho \subseteq \pi$ to state that a vertex $v$ is a member of the sequence representing $\pi$ and that a path $\rho$ is a consecutive subsequence of $\pi$, respectively.

As mentioned in Section I, we study in this paper a greedy approach. The idea, explained formally further, is as follows. We associate with every white vertex (server) an attribute called *rank*. It denotes how many times the server was reallocated, i.e., how many times was the corresponding white vertex a member of an augmenting path. We wish to minimize the maximum rank over all white vertices. A *rank of a path* $\pi$ is defined to be equal to the maximum rank among the white vertices on $\pi$. In every turn $t$, if $b_t$ can be matched, our algorithm chooses an augmenting path $b_t \xrightarrow{\pi_t} s \in S_{t-1}$ such that every suffix of $\pi_t$ is of minimum rank.

We now introduce this idea formally. We define the rank function on $W$ for every turn $t$. We set $\text{rank}_0(w) = 0$ for every $w \in W$ (it is always the case that $\text{rank}_t(w) = 0$ iff $w \in S_t$, i.e., $w$ is a seed in turn $t$). The $\text{rank}_t$ function is obtained from $\text{rank}_{t-1}$ by increasing by one the rank of every white vertex $w \in \pi_t$:

$$\text{rank}_t(w) = \begin{cases} \text{rank}_{t-1}(w) & \text{if } w \notin \pi_t, \\ \text{rank}_{t-1}(w) + 1 & \text{if } w \in \pi_t. \end{cases}$$

We define the rank of a path $\pi$ in $G_t$ (or in $G_{t+1}^{\text{int}}$) as

$$\text{rank}_t(\pi) = \max_{w \in \pi} \text{rank}_t(w).$$

In addition to the rank function on the white vertices we introduce a *tier* function on the black vertices. The *tier* of a black vertex is the minimal rank of a directed path in $G_t^{\text{int}}$ leading to a seed. For any $t \in [n]$ and $b \in B_t$ we define

$$\text{tier}_t(b) = \begin{cases} \infty & \text{if there is no path from } b \text{ to } S_t, \\ \min_{s \in S_t, (b \xrightarrow{\pi} s) \subseteq G_t} \text{rank}_t(\pi) & \text{otherwise.} \end{cases}$$

Observe that for every white vertex $w$ there is at most one outgoing edge in every $G_t$ (i.e., the matched edge) and denote the black vertex matched with $w$ by $b_w^t$ (we skip the upper index when it is clear from the context). Each edge $wb_w^t$ in $G_t$ from a white vertex $w$ to the black vertex $b_w^t$ is *tired* in $G_t$. Moreover, edge $bw$ is tiered if the tier of $b$ is not smaller than the tier of $b_w^t$ and the rank of $w$. The set of all tiered edges at the beginning of turn $t$ is denoted by $E_t^\star \subseteq E_t^{\text{int}}$,

$$E_t^\star = \left\{bw \in E_t^{\text{int}} : b \in B,\ w \in W \text{ and } \operatorname{tier}_{t-1}(b) \geqslant \max\left\{\operatorname{tier}_{t-1}(b_w), \operatorname{rank}_{t-1}(w)\right\}\right\} \cup \left\{wb_w^{t-1} \in E_t^{\text{int}} : w \in W\right\}.$$

This definition is extended to paths by saying that a path is *tiered* if it is a simple path that consists entirely of tiered edges.

In each round $t$, if possible, our algorithm chooses in $G_t^{\text{int}}$ a tiered path $b_t \xrightarrow{\pi_t} s \in S_{t-1}$. The definition of a tiered path enforces that each suffix of $\pi_t$ is optimal in a sense that it minimizes the rank. The algorithm then changes the orientation of the edges on $\pi_t$ and increases the ranks of the white vertices. We refer to any such algorithm as a *tiered* algorithm. In the next sections we first show several properties of all tiered algorithms, and only after that we show how exactly our algorithm efficiently chooses the paths.

To conclude this section, we introduce a structure that will become useful in many of our proofs later. This structure also explains why the unmatched white vertices are called seeds. A *seeded tree* $\tau$ is any directed tree rooted at some seed $s \in S_t$. Any forest of $|S_t|$ vertex-disjoint seeded trees that spans all vertices of $G$ that can reach a seed is called *seeded*. We call a seeded forest *tiered* if it consists entirely of tiered edges. A *seeded path* is any simple path to a seed.

Observe that not all vertices are contained in some seeded tree. The algorithm may produce orientations in which some vertices cannot reach any seed. We essentially ignore these vertices, as there is no need for the algorithm to visit them. Once a vertex cannot reach any seed, it will never be able to reach a seed again. Indeed, consider the set of vertices reachable from such a vertex. This set contains no seeds. Moreover, all the edges between this set and the rest of vertices are directed towards this set. Hence, no alternating path can reorient the edges between the set and the remaining part of the graph. We refer to vertices that cannot reach a seed as *dead*, the remaining vertices are called *alive*. As a consequence of the above argument, it only makes sense to consider the turns in which $b_t$ can be matched. In the remaining turns neither the alive part of the graph changes, nor do the ranks. Thus, from now on, we completely ignore the black vertices that cannot be matched and the turn index $t$ corresponds to the turns where the augmenting path can be found.

## III. Ranks and tiers

In this section we present basic properties of ranks and tiers when they are maintained by tiered algorithms. These results will form the basis for the following sections. We start this section by proving that tiers of vertices do not decrease. Next, we prove that vertices with high tiers are far from the seeds. To be more precise, we show that for a vertex $b$ of finite tier, the length of the shortest path from $b$ to a seed is at least its tier. Let us start with the following observations.

**Observation III.1.** The following facts hold for $t \in [n]$:

1) There exists a tiered path from any alive vertex $b \in B_t$ to a seed in $S_t$.
2) For any tiered path $\pi$ from $b \in B_t$ to a seed $s \in S_t$ it holds that $\operatorname{tier}_t(b) = \operatorname{rank}_t(\pi)$.
3) There exists a seeded tiered forest in $G_t$.
4) If there is a non-simple directed path from $b$ to $w$ in $G_t$, then there is also a simple directed path from $b$ to $w$ with the same or smaller rank.

We are ready to prove the following important lemma.

**Lemma III.2.** *For any vertex $b \in B_t$ it holds that* $\operatorname{tier}_t(b)$ *is a non-decreasing function of $t$.*

*Proof.* Fix some seeded tiered forest $\psi$ in $G_t$, which exists by Observation III.1.1. First assume that $b$ is alive, so $b \in B(\psi)$. Let $\operatorname{dist}_\psi(b)$ denote the black distance from $b$ to a seed in $\psi$, i.e., the number of black vertices on a path from $b$ to the corresponding seed (not including $b$). We proceed by induction on $\operatorname{dist}_\psi(b)$. The vertices with $\operatorname{dist}_\psi(b) = 0$ can directly reach seeds, which means that $\operatorname{tier}_t(b) = 0$. A non-seed cannot turn into a seed, so $\operatorname{tier}_{t-1}(b) = 0$ for such vertices.

Let $b$ be an arbitrary alive vertex of $B_t$. Let $\pi \subseteq \psi$ be (the only) tiered path in $\psi$ connecting $b$ with a seed. Let $w$ be a direct successor of $b$ on $\pi$ and $b'$ be a black successor of $w$ on $\pi$ (the case when $w$ is a seed was already covered). Since $\operatorname{dist}_\psi(b') < \operatorname{dist}_\psi(b)$, by induction we have $\operatorname{tier}_t(b') \geqslant \operatorname{tier}_{t-1}(b')$. By Observation III.1.2, since $\pi$ is tiered, it holds that $\operatorname{tier}_t(b) = \operatorname{rank}_t(\pi) = \max\{\operatorname{rank}_t(w), \operatorname{tier}_t(b')\}$. Denote by $\rho$ the path along which the edges were reversed in turn $t$. We distinguish three cases shown in Figure III.1.

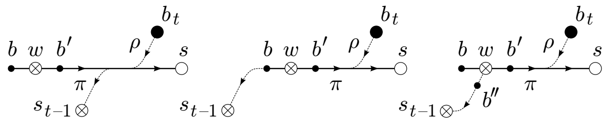

Fig. III.1. Three cases depending the location of vertex $b$ with regard to the augmenting path $\rho$.

If $w \notin \rho$, then $bwb'$ is a directed path in $G_{t-1}$, so

$$\begin{aligned}\operatorname{tier}_t(b) &\geqslant \max\left\{\operatorname{rank}_t(w), \operatorname{tier}_t(b')\right\}\\ &\geqslant \max\left\{\operatorname{rank}_{t-1}(w), \operatorname{tier}_{t-1}(b')\right\}\\ &\geqslant \operatorname{tier}_{t-1}(b).\end{aligned}$$

The last inequality follows from Observation III.1.4.

If $b'wb \subseteq \rho$ (i.e., the path $bwb'$ changed its orientation), since $\rho$ is a tiered path in turn $t-1$, it holds that $\operatorname{tier}_{t-1}(b') \geqslant \operatorname{tier}_{t-1}(b)$. Therefore

$$\operatorname{tier}_t(b) \geqslant \operatorname{tier}_t(b') \geqslant \operatorname{tier}_{t-1}(b') \geqslant \operatorname{tier}_{t-1}(b).$$

Finally, it might be that $b'wb'' \subseteq \rho$ where $b'' \in B_t, b'' \neq b$. That means that there is a path with prefix $bwb''$ to a seed

$s \in S_{t-1}$ in $G_{t-1}$. On the other hand the smallest rank of the path from $b$ to a seed in $G_{t-1}$ equals $\mathrm{tier}_{t-1}(b)$, so

$$\max\{\mathrm{rank}_{t-1}(w), \mathrm{tier}_{t-1}(b'')\} \geqslant \mathrm{tier}_{t-1}(b). \tag{1}$$

Because $b'$ lies on the tiered path (the fragment of tiered forest $\psi$) from $b$ to a seed in $G_t$ and $b''$ lies on the tiered path (the fragment of augmenting path $\rho$) from $b'$ to a seed in $G_{t-1}$ we obtain that

$$\mathrm{tier}_t(b) \geqslant \mathrm{tier}_t(b') \geqslant \mathrm{tier}_{t-1}(b') \geqslant \mathrm{tier}_{t-1}(b''). \tag{2}$$

Taking together (1) and (2) we obtain that

$$\mathrm{tier}_t(b) \geqslant \max\{\mathrm{rank}_t(w), \mathrm{tier}_{t-1}(b'')\} \geqslant \mathrm{tier}_{t-1}(b).$$

The case when $b$ is a dead vertex, so its tier becomes or remains infinite, completes the proof. □

The next lemma shows, that high ranks are followed by high tiers. It directly implies, that vertices of high tiers are far from the seeds.

**Lemma III.3.** *For any white vertex $w \in W \setminus S_t$, we have $\mathrm{tier}_t(b^t_w) + 1 \geqslant \mathrm{rank}_t(w)$.*

*Proof.* Assume that $\mathrm{rank}_t(w) > 0$ (i.e., $w \notin S_t$) and let $\tilde{t}$ be the last time the rank of $w$ changed, i.e.,

$$\tilde{t} = \max\{i \in [t] : \mathrm{rank}_{i-1}(w) < \mathrm{rank}_i(w)\}.$$

Then, since the augmenting path $\pi_{\tilde{t}}$ was tiered, it holds that $\mathrm{tier}_{\tilde{t}-1}(b^t_w) \geqslant \mathrm{rank}_{\tilde{t}-1}(w)$. However, by Lemma III.2, the tier does not decrease, so

$$\mathrm{tier}_t(b^t_w) + 1 \geqslant \mathrm{tier}_{\tilde{t}-1}(b^t_w) + 1 \geqslant$$
$$\geqslant \mathrm{rank}_{\tilde{t}-1}(w) + 1 = \mathrm{rank}_t(w).$$

□

**Corollary III.4.** Let $B_t \ni b \xrightarrow{\pi} s \in S_t$. Every white vertex $w \in \pi$ of rank $\mathrm{rank}_t(w) \geqslant 1$ is followed by some vertex $w'$ of rank $\mathrm{rank}_t(w') \geqslant \mathrm{rank}_t(w) - 1$.

*Proof.* By Lemma III.3 $\mathrm{tier}_t(b_w) \geqslant \mathrm{rank}_t(w) - 1$. By definition of tier, any path (including $\pi$) from $b_w$ to any seed contains a white vertex $w'$ such that $\mathrm{rank}_t(w') \geq \mathrm{tier}_t(b_w)$. □

**Corollary III.5.** The length of the shortest path from $b \in B_t$ to a seed is at least $2\,\mathrm{tier}_t(b) + 1$.

## IV. Upper bound on white vertex usage

In this section we prove the main result of the paper, which gives an $O(\sqrt{n})$ upper bound on the ranks of the vertices generated by any tiered algorithm. The proof proceeds in two stages. Lemma IV.1 captures a certain reversed behavior of a tiered algorithm crucial for our proof, pictured by Figure IV.1. Figure IV.1 presents the state of the graph after (to the left: graph $G_t$) and before (to the right: graph $G_{t-1}$) the augmentation step. Assume that graph $G_t$ contains seeded paths $\mu_1 \ldots \mu_l$. On the picture to the left, the ranks of these paths are shown by the heights of the corresponding bars, sorted by rank. Lemma IV.1 states, that before the augmentation step (the right side of the picture), each path $\mu_i$ had its counterpart $\lambda_i$, plus there was one more path $\rho$. The interesting part is that the heights of the bars (the ranks of the corresponding paths) decrease at most by one, and the newly added bar (the one corresponding to $\rho$) dominates the bars that decrease.

Hence, the line inclined at the angle of 45 degrees through the right top point of the bar representing $\lambda_l$ does not drop: whenever the bars reduce their height by one, an additional bar appears to support the line at its initial position. The idea

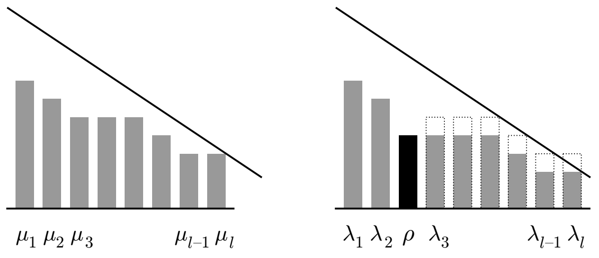
$\mu_1\ \mu_2\ \mu_3$
$\mu_{l\text{-}1}\ \mu_l$
$\lambda_1\ \lambda_2\ \rho\ \lambda_3$
$\lambda_{l\text{-}1}\ \lambda_l$

Fig. IV.1. The bars corresponding to the paths after and before the augmentation step.

is that for any white vertex whose rank is $r$ at some point, there is a turn when a seeded path of rank at least $r$ exists. So, if the algorithm produces a vertex of rank $r$, we start with one bar of height $r$ and reverse the steps of the algorithm. We reach a point when there is a linear in $r$ number of vertex disjoint paths whose height is also linear in $r$. The next lemmas formally prove our result.

It is worth mentioning that in the following reasoning it is enough to consider alive vertices. Dead vertices do not affect the definition of $\mathrm{rank}_t$ and $\mathrm{tier}_t$ and behavior of the algorithm.

**Lemma IV.1.** *Let $\mu_1, \mu_2, \ldots, \mu_l$ be a sequence of vertex-disjoint seeded paths in $G_t$. Also, let $b_t \xrightarrow{\pi_t} s$ be the augmenting path in turn t, i.e., in $G^{\mathrm{int}}_t$. Then, there exist vertex-disjoint seeded paths $\lambda_1, \ldots, \lambda_l$ and $\rho$ in $G_{t-1}$ such that*

$$\mathrm{rank}_{t-1}(\pi) \leqslant \mathrm{rank}_{t-1}(\rho), \tag{3}$$

$$\mathrm{rank}_t(\mu_i) \leqslant \mathrm{rank}_{t-1}(\lambda_i) \quad \textit{for } \mathrm{rank}_t(\mu_i) > \mathrm{rank}_{t-1}(\pi) + 1, \tag{4}$$

$$\mathrm{rank}_t(\mu_i) \leqslant \mathrm{rank}_{t-1}(\lambda_i) + 1 \quad \textit{otherwise.} \tag{5}$$

*Proof.* Let $w_i$ be the highest-ranked white vertex of $\mu_i$ that is the closest to the seed. Without loss of generality we can assume that $\mu_i$ starts with $w_i$. Then, for $i \in [l]$, directed path $\mu_i$ is of the form $W \ni w_i \xrightarrow{\mu_i} s_i \in W$ where $s_i \in S_t$ is a seed.

Now, we define a 0/1-flow network $F$ where $V(F) = V(G^{\mathrm{int}}_t) \cup \{\varsigma, \tau\}$ and

$$E(F) = E(G^{\mathrm{int}}_t) \cup \{\varsigma b_t, \varsigma w_1, \ldots, \varsigma w_l\} \cup \{s\tau, s_1\tau, \ldots, s_l\tau\}$$

where $\varsigma$ and $\tau$ are artificially added source and sink respectively (see Fig. IV.2).

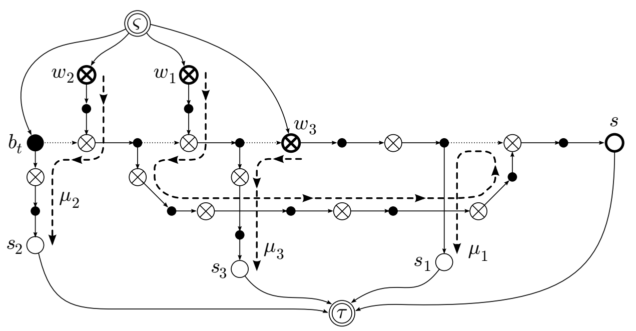

Fig. IV.2. Example of a flow network $F$, where solid arcs form the flow from source $\varsigma$ to sink $\tau$ with the total value 4.

Because $b_t \xrightarrow{\pi_t} s$ is an augmenting path expanding matching in graph $G_t^{\text{int}}$, we have that $\varsigma b_t \xrightarrow{\pi_t} s\tau$ is a path which defines a flow $f_1$ in $F$ with the total flow value equal to 1. Moreover, we obtain that the residual graph $F^{f_1}$ in restriction to $V(G_t)$ has the same orientation as $G_t$. Because $\mu_1, \ldots, \mu_l$ are pairwise vertex-disjoint directed paths in $G_t$, we have that paths $\mu_i$ of the form $\varsigma w_i \xrightarrow{\mu_i} s_i\tau$ are pairwise edge-disjoint directed paths in residual graph $F^{f_1}$. Consecutive application of the augmenting paths $\mu_1, \ldots, \mu_l$ expands the flow $f_1$ to a flow $f_{l+1}$ with the total value of $l+1$.

The flow $f_{l+1}$ determines $l+1$ edge-disjoint paths in $F$ from $\varsigma$ to $\tau$ as $f_{l+1}$ is a 0/1-flow. Because $b_t, w_1, \ldots, w_l$ are all neighbors of $\varsigma$ and $s_1, \ldots, s_l, s_{l+1} = s$ are all neighbors of $\tau$, we obtain $l+1$ paths of the form $\varsigma b_t w_{l+1} \xrightarrow{\rho} s_{\sigma(l+1)}\tau$ and $\varsigma w_i \xrightarrow{\lambda_i} s_{\sigma(i)}\tau$ for $i \in [l]$ where $\sigma$ is some permutation of $[l+1]$ and $\lambda_1, \ldots, \lambda_l, \rho$ are some edge-disjoint directed paths in $G_t^{\text{int}} \subseteq F$. Because $w_1, \ldots, w_l, b_t$ are different and $s_1, \ldots, s_l, s_{l+1}$ are also different and each vertex in $G_{t-1}$ has out-degree or in-degree equals at most 1, paths $\lambda_1, \ldots, \lambda_l, \rho$ are also vertex-disjoint.

To see (3), it is enough to mention that path $\pi$ was chosen in turn $t$ so it was the smallest rank seeded path from $b_t$. Because the augmenting path $\pi$ in turn $t$ uses white vertices with rank at most $\text{rank}_{t-1}(\pi)$, lines (4)-(5) are obvious. □

The next lemma presents the core of our result. Imagine that for some white vertex $w$ its rank was raised to $\text{rank}_{\tilde{t}}(w)$ by some turn $\tilde{t}$. The lemma states that we will find a turn before turn $\tilde{t}$, when a collection of vertex-disjoint paths existed, such that the sum of their ranks was at least quadratic with respect to $\text{rank}_{\tilde{t}}(w)$. The proof heavily bases on Lemma IV.1, which allows us to reverse the steps of the algorithm.

We begin the reversing process in turn $t'$, where a path $\rho$ exists with rank at least $\text{rank}_{\tilde{t}}(w) - 1$. Starting from a one element collection $\{\rho\}$, we iteratively construct a collection of paths using Lemma IV.1. From the collection in step $t$ we obtain a collection in step $t-1$. There are two important invariants in this reversing process which imply the lemma:

- minimum rank in the collection decreases at most by one
- if the minimum rank in the collection decreases, the size of the collection grows by one
- there is a moment $t_0$ when all ranks in the collection are equal to 0

Somewhere before $t_0$ there is a moment, where there are sufficiently many paths of sufficiently high ranks, as illustrated in Figure IV.3. Again, each bar represents a path and the heights of the bars are the ranks of the corresponding paths. Let us now move on to formalizing this idea.

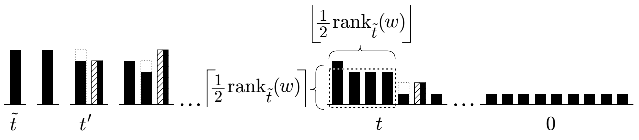

Fig. IV.3. In time $t$ there is a collection of $\lfloor \text{rank}_{\tilde{t}}(w)/2 \rfloor$ paths of rank at least $\lceil \text{rank}_{\tilde{t}}(w)/2 \rceil$.

**Lemma IV.2.** *For any turn $\tilde{t} \in [n]$ and any white vertex $w \in W_{\tilde{t}}$ there is $t \leqslant \tilde{t}$ and $j$ pairwise vertex-disjoint seeded simple paths $P_1, \ldots, P_j$ in $G_t$ such that*

$$\left\lfloor \frac{1}{4}\text{rank}_{\tilde{t}}(w)^2 \right\rfloor \leqslant \text{rank}_t(P_1) + \ldots + \text{rank}_t(P_j).$$

*Proof.* Let $t' \leqslant \tilde{t}$ be the round when the rank of white vertex $w$ was changed the last time before round $\tilde{t}$. Now, we define sequence $B_{t'-1}, \ldots, B_0$, where $B_k$ is a collection of edge-disjoint simple seeded paths in $G_k$ for $k = 0, \ldots, t'$. From the definition of $t'$ we know that $\text{rank}_{\tilde{t}}(w) = \text{rank}_{t'}(w)$ and $\text{rank}_{t'-1}(\pi_{t'}) \geqslant \text{rank}_{t'-1}(w)$. If we remove the vertex $b_{t'}$ from $\pi_{t'}$, we obtain a simple path $\rho$ in $G_{t'-1}$ with rank

$$\begin{aligned}\text{rank}_{t'-1}(\rho) = \text{rank}_{t'-1}(\pi_{t'}) \geqslant \text{rank}_{t'-1}(w) = \\ = \text{rank}_{t'}(w) - 1 = \text{rank}_{\tilde{t}}(w) - 1.\end{aligned}$$

Thus, we define $B_{t'-1} = \{\rho\}$. Now, let us assume that $B_k = \{\mu_1, \ldots, \mu_l\}$ is defined for some $k \in \{1, \ldots, t'-1\}$. Again, by applying Lemma IV.1 to $G_k$, a sequence of paths $\mu_1, \ldots, \mu_l$, and augmenting path $\pi_k$, we obtain simple seeded paths $\lambda_1, \ldots, \lambda_l, \overline{\rho}$ in $G_{k-1}$ satisfying conditions (1)-(3). Then we define $B_{k-1}$ as

$$B_{k-1} = \begin{cases} \{\lambda_1, \ldots, \lambda_l, \overline{\rho}\} & \exists_{i \in [l]}\, \text{rank}_k(\mu_i) > \text{rank}_{k-1}(\lambda_i), \\ \{\lambda_1, \ldots, \lambda_l\} & \\ & \text{otherwise (i.e., } \forall_{i \in [l]}\, \text{rank}_k(\mu_i) = \text{rank}_{k-1}(\lambda_i).) \end{cases}$$

If $\text{rank}_k(\mu_i) > \text{rank}_{k-1}(\lambda_i)$ for some $i \in [l]$ then $\text{rank}_k(\mu_i) \leqslant \text{rank}_{k-1}(\pi_k) + 1 \leqslant \text{rank}_{k-1}(\overline{\rho}) + 1$, by conditions (1)-(3) from Lemma IV.1. Again, applying inequalities (1)-(3) from Lemma IV.1, we obtain that

$$\min_{\lambda \in B_{k-1}} \text{rank}_{k-1}(\lambda) \geqslant \begin{cases} \min_{\mu \in B_k} \text{rank}_k(\mu) - 1 \\ \qquad \text{if } |B_{k-1}| - |B_k| = 1, \\ \min_{\mu \in B_k} \text{rank}_k(\mu) \\ \qquad \text{otherwise (i.e., } |B_{k-1}| - |B_k| = 0). \end{cases}$$

Multiple use of the above inequality gives us

$$\begin{aligned}\min_{\lambda\in B_k}\operatorname{rank}_k(\lambda) &\geqslant \min_{\mu\in B_{t'-1}}\operatorname{rank}_{t'-1}(\mu)-(|B_k|-1)\\ &=\operatorname{rank}_{\tilde t}(w)-|B_k|\,,\end{aligned}\tag{6}$$

for any chosen $k = 0,\dots,t'-1$. The formula (6) implies that family $B_0$ is the size of at least $\operatorname{rank}_{\tilde t}(w)$ by $\operatorname{rank}_0(\lambda) = 0$ for all $\lambda \in B_0$. Thus, we can choose $t \in [t']$ for which $|B_t| = \left\lfloor \frac{1}{2}\operatorname{rank}_{\tilde t}(w)\right\rfloor$ and by (6) we have $\min_{\lambda\in B_t}\operatorname{rank}_t(\lambda) \geqslant \left\lceil \frac{1}{2}\operatorname{rank}_{\tilde t}(w)\right\rceil$. Finally, we obtain the family $B_t = \{P_1,\dots,P_j\}$ of size $j = |B_t| = \left\lfloor \frac{1}{2}\operatorname{rank}_{\tilde t}(w)\right\rfloor$ where each $P_i \in B_t$ has the rank at least $\left\lceil \frac{1}{2}\operatorname{rank}_{\tilde t}(w)\right\rceil$. This together gives a desired lower bound on the sum of the ranks of $P_i$:

$$\begin{aligned}\left\lfloor\frac{1}{4}\operatorname{rank}_{\tilde t}(w)^2\right\rfloor &\leqslant \left\lfloor\frac{1}{2}\operatorname{rank}_{\tilde t}(w)\right\rfloor\cdot\left\lceil\frac{1}{2}\operatorname{rank}_{\tilde t}(w)\right\rceil\\ &\leqslant |B_t|\cdot\min_{P_i\in B_t}\operatorname{rank}_t(P_i)\\ &\leqslant \operatorname{rank}_t(P_1)+\ldots+\operatorname{rank}_t(P_j).\end{aligned}$$

□

Now, we are ready to prove the main theorem:

**Theorem IV.3.** *For any tiered algorithm it holds that* $\operatorname{rank}_n(w) < \sqrt{2n}$ *for every* $w \in W$.

*Proof.* By Lemma IV.2 there is round $t \in [n]$ in which there is a collection of $j$ pairwise vertex-disjoint directed simple paths $P_1,\dots,P_j \in G_t$ for which the sum of their ranks is $\left\lfloor\frac{1}{4}\operatorname{rank}_n(w)^2\right\rfloor$. This gives that $P_1,\dots,P_j$ are paths of total length greater than $\frac{1}{2}\operatorname{rank}_n(w)^2$ by Corollary III.5. On the other hand, graph $G_t$ has at most $n$ vertices, so $\operatorname{rank}_n(w) < \sqrt{2n}$. □

## V. The Algorithm

In the previous sections we showed that any tiered algorithm rematches a white vertex at most $O(\sqrt{n})$ times. This shows that the sum of the lengths of all used augmenting paths is bounded by $O(n\sqrt{n})$. Unfortunately the algorithm not only needs to apply the augmenting paths, but also needs to find them, so the total running time might be much higher than this bound. In this section we show an incremental algorithm finding maximal matching with a total running time of $O(m\sqrt{n})$, which is equal to the running time of the Hopcroft–Karp algorithm.

In essence, we observe that we can pay for the search by increasing the ranks of the visited white vertices, and the bound of $O(\sqrt{n})$ for the white vertex usage still holds.

Algorithm *Match* proposed here proceeds in the following way (see Algorithm 1 for the pseudocode). At the initialization time, *Match* receives $G_0$ as an input, and sets $w.rank^* = 0$ for every white vertex $w \in W$. We denote the ranks produced by *Match* in turn $t$ as $\operatorname{rank}^*_t(w)$. So, due to the initialization step, for every white vertex $w \in W$ we have $\operatorname{rank}^*_0(w) = 0$. The tier function for every $t \in [n]$ is defined, as before, based on the rank function, i.e., for any $b \in B_t$

$$\operatorname{tier}^*_t(b) = \begin{cases}\infty & \text{if there is no path from } b \text{ to } S_t,\\ \min_{s\in S_t, b\xrightarrow{\pi}s}\operatorname{rank}^*_t(\pi) & \text{otherwise.}\end{cases}$$

In turn $t$, the algorithm searches for the tiered augmenting path $\pi_t$ starting at $b_t$. It proceeds recursively. The recursive function $\textsc{Search}(w)$ is called for a white vertex $w$, initially equal to the white neighbor of $b_t$ with the smallest rank. If $w$ is a seed ($\textsc{Unmatched}(w)$ returns `true`) then $w$ increases its rank and, since a seeded path is found, $\textsc{Search}(w)$ returns true. Otherwise the recursive step is determined by the black vertex $b^{t-1}_w$ (matched with $w$). The algorithm (using function $\textsc{SmallestNeighborUnmatchedTo}(b^{t-1}_w)$) picks the white neighbor $w'$ of $b^{t-1}_w$ with the minimum rank, i.e.,

$$w'.rank^* = \min_{w''\;:\;b^{t-1}_w w''\in E^{\text{int}}_t} w''.rank^*.$$

The value of $w'.rank^*$ is a lower bound for $\operatorname{tier}^*_{t-1}(b^{t-1}_w)$. A successful recursive call on $w'$ recursively finds a directed path to a seed with rank bounded from above by $w'.rank^*$. Hence, if the call is successful, $\operatorname{tier}^*_{t-1}(b^{t-1}_w) = w'.rank^*$. This certifies that both edges $wb^{t-1}_w$ and $b^{t-1}_w w'$ are tiered and so the algorithm found a tiered path. In such case the recursion terminates returning `true` and the ranks are increased along the tiered path that was found. Each time a rank of a white vertex $w$ is increased, the function $\textsc{UpdateInformation}(b, w)$ is called in order to update $w.rank^*$ for all the black vertices $b$ that can directly reach $w$. An unsuccessful recursive call on $w'$ certifies, that $\operatorname{tier}^*_{t-1}(b_{w'}) > w'.rank^*$. On such an event, the algorithm increases $w'.rank^*$ and continues picking the neighbors of $b^{t-1}_w$ of minimum rank and calling the recursion on them. This process finishes when either the call is successful or the only available ranks reach the bound of $\sqrt{2n}$. In the latter case the algorithm located a dead region, which never will be entered again. Note that every time a white vertex is visited by the algorithm, its rank increases. We soon show, that unless a vertex is dead, its rank is bounded by $\sqrt{2n}$.

As for the running time, note that the function $\textsc{UpdateInformation}(b, w)$ can be handled in constant time for every neighbor $b$ pointing at $w$. For example each black vertex can keep a list of lists of white vertices. This allows to update the information in this recursive call in the time bounded by the degree of $w$. At the same time the rank of $w$ is increased, and this happens at most $O(\sqrt{n})$ times for any white vertex. The total running time of $O(m\sqrt{n})$ follows.

Consider an arbitrarily chosen vertex $w$. We distinguish two phases of the algorithm *Match* in turn $t$, encoded as $\textsc{Match}(b_t)$ in Algorithm 1. The first phase consits of a sequence (which can be empty) of calls to $\textsc{Search}(w)$ which return `false`. Each such call does not rematch $w$ but increases $w.rank^*$. The second phase consists of at most one call $\textsc{Search}(w)$ with returns `true`. In such call, $\textsc{Search}(w)$ rematches vertex $w$ and increases $w.rank^*$ as well. We formulate the following lemma, which can be proven by induction on the number of calls to $\textsc{Search}(\cdot)$. The lemma describes the difference between the above mentioned two phases.

**Lemma V.1.** *Consider any recursive call* $\textsc{Search}(w)$ *originating from a call to* $\textsc{Match}(b_t)$ *for any* $w \in W$. *There is a path* $\rho$ *from* $w$ *to some seed with ranks on white vertices not greater then* $w.rank^*$ *if and only if* $\textsc{Search}(w)$ *returns* `true`

**Algorithm 1** Algorithm *Match*

```
procedure MATCH($b_t$)
    $w' \leftarrow$ SMALLESTNEIGHBORUNMATCHEDTO($b_t$)
    while $w'.rank^* < \sqrt{2n}$ do
        if SEARCH($w'$) then
            $match \leftarrow match \cup \{(b_t, w')\}$
            return true
        $w' \leftarrow$ SMALLESTNEIGHBORUNMATCHEDTO($b_t$)
    return false

procedure SEARCH($w$)
    $w.rank^* \leftarrow w.rank^* + 1$
    for each $b$ such that $bw \in E_t^{\text{int}}$ do
        UPDATEINFORMATION($b, w$)
    if UNMATCHED($w$) then
        return true
    $w' \leftarrow$ SMALLESTNEIGHBORUNMATCHEDTO($b_w$)
    while $w'.rank^* < w.rank^*$ do
        if SEARCH($w'$) then
            $match \leftarrow match \setminus \{(w, b_w)\} \cup \{(b_w, w')\}$
            return true
        $w' \leftarrow$ SMALLESTNEIGHBORUNMATCHEDTO($b_w$)
    return false
```

*and rematches edges along some path $\pi$ from $w$ to some seed with ranks on white vertices smaller or equal to $w.rank^*$.*

One of the consequences of the above lemma is that during the first phase, although $w.rank^*$ grows, it does not become greater than $\text{tier}^*_{t-1}(b_w^{t-1})$. Let us define $rank^{\circledast}_{t-1}(w)$ as a value $w.rank^*$ immediately after the first phase. It holds that

$$\begin{aligned}\text{if } rank^{\circledast}_{t-1}(w) > \text{rank}^*_{t-1}(w) \\ \text{then } rank^{\circledast}_{t-1}(w) \leqslant \text{tier}^*_{t-1}(b_w^{t-1})\end{aligned} \tag{7}$$

and so we obtain the following:

**Observation V.2.** For any $t \in [n]$ and any path $w \xrightarrow{\pi} s$, $w \in W, s \in S_{t-1}$, it holds that $rank^{\circledast}_{t-1}(\pi) = \text{rank}^*_{t-1}(\pi)$.

Now if $b_t$ is dead, procedure MATCH($b_t$) tries to match it anyway. Due to Observation V.2, all vertices visited during such a call to MATCH($b_t$) incur only the first phase, and as a consequence the ranks of the paths do not change. This allows us to simplify the description and omit the rounds when dead vertices appear.

With respect to the second phase, we can observe that during that phase the algorithm increases by one ranks of white vertices on the augmenting path $\pi_t$, i.e.,

$$\text{rank}^*_t(w) = \begin{cases} rank^{\circledast}_{t-1}(w) & \text{if } w \notin \pi_t, \\ rank^{\circledast}_{t-1}(w) + 1 & \text{if } w \in \pi_t. \end{cases} \tag{8}$$

By Observation V.2 and equation (8) very similar arguments as for Lemma III.2 and Lemma III.3 prove the corresponding lemmas:

**Lemma V.3.** *For any vertex $b \in B_t$ it holds that $\text{tier}^*_t(b)$ is a non-decreasing function of $t$.*

**Lemma V.4.** *For any white vertex $w \in W \setminus S_t$, we have $\text{tier}^*_t(b_w) + 1 \geqslant \text{rank}^*_t(w)$.*

The next two corollaries follow.

**Corollary V.5.** Let $b \xrightarrow{\pi} s$, for a black vertex $b$, be a path that ends at a seed $s$. Every white vertex $w \in \pi$ of rank $\text{rank}^*_t(w) \geqslant 1$ is followed by some vertex $w'$ of rank $\text{rank}^*_t(w') \geqslant \text{rank}^*_t(w) - 1$.

**Corollary V.6.** The length of the shortest path from a black vertex $b$ to a seed is at least $\text{tier}^*_t(b)$.

We move on to the two lemmas crucial for our approach.

**Lemma V.7.** *Let $\mu_1, \mu_2, \ldots, \mu_l$ be a sequence of vertex-disjoint seeded simple paths to distinct seeds in $G_t$. Also, let $b_t \xrightarrow{\pi_t} s$ be the augmenting path in turn $t$, i.e., in $G^{int}_{t-1}$. Then, there exist vertex-disjoint simple seeded paths $\lambda_1, \ldots, \lambda_l$ and $\rho$ in $G_{t-1}$ to distinct seeds such that*

$$\text{rank}^*_{t-1}(\pi_t) \leqslant \text{rank}^*_{t-1}(\rho), \tag{9}$$

$$\text{rank}^*_t(\mu_i) \leqslant \text{rank}^*_{t-1}(\lambda_i) \quad \textit{for } \text{rank}^*_t(\mu_i) > \text{rank}^*_{t-1}(\pi) + 1, \tag{10}$$

$$\text{rank}^*_t(\mu_i) \leqslant \text{rank}^*_{t-1}(\lambda_i) + 1 \quad \textit{otherwise.} \tag{11}$$

*Idea of proof.* The analogous argument as in proof of Lemma IV.1 implies that the paths $\lambda_1, \ldots, \lambda_l$ and $\rho$ as described exist in $G^{\text{int}}_t$. Due to Observation V.2 and the equation (8) we obtain the desired inequalities. □

Now, the proof of Lemma IV.2 directly translates into an argument for the following lemma:

**Lemma V.8.** *For any turn $\tilde{t} \in [n]$ and any white vertex $w \in W_{\tilde{t}}$ there is $t \leqslant \tilde{t}$ and $j$ pairwise vertex-disjoint seeded simple paths $P_1, \ldots, P_j$ in $G_t$ such that*

$$\left\lfloor \frac{1}{4} \text{rank}^*_t(w)^2 \right\rfloor \leqslant \text{rank}^*_t(P_1) + \ldots + \text{rank}^*_t(P_j).$$

As a consequence, we obtain a corollary analogous to Theorem IV.3.

**Corollary V.9.** For any tiered algorithm it holds that $\text{rank}^*_n(w) < \sqrt{2n}$ for every $w \in W$.

## VI. Applications

In this section we describe few applications of our result. First, we present a linear time incremental approximation algorithm which, for any constant $\varepsilon$, maintains a $(1 - \varepsilon)$-approximation of maximum matching in a bipartite graph revealed online. Interestingly, if we settle for approximation, we do not need Theorem IV.3 for the algorithm to work. This algorithm implies a simple offline algorithm with a simple analysis, also not requiring Theorem IV.3. It shows that the real value of our result lies in the online nature of the problem.

Further, we reduce the decremental version of the problem to the incremental version, both in exact and approximate case. We finish the section by generalizing our result to the weighted setting.

### A. Incremental approximation and offline analysis

The approximation algorithm is based on the following well known fact, that a matching for which all augmenting paths are long is a good approximation of an optimum matching in a bipartite graph.

**Claim VI.1.** *Let $M^*$ be an optimum matching in a bipartite graph $B$, and $M$ be a matching. If the length of the shortest path augmenting $M$ is at least $k$, then $|M| \geqslant |M^*|(1-\frac{2}{k})$.*

Combined with the above claim, our algorithm gives an efficient $(1-\varepsilon)$-approximation for incremental bipartite matching. We try to match a black vertex $b_t$ as long as $\min_{w\,:\,b_t w\,\in\,E_t^{\text{int}}} \operatorname{rank}(w) \leq k$ for some constant $k$. Once this value is higher than $k$, we leave $b_t$ unmatched. In every turn, due to Lemma III.5, all augmenting paths starting at vertices we left out are of length at least $k$. Hence, in each turn we maintain a $(1-\frac{2}{k})$-approximation of the optimum matching. Moreover, since we took care that ranks are bounded by $k$, the total time used by the approximation algorithm is $O(km)$. We summarize this interesting observation in the following lemma. It is worth emphasizing that we do not need Theorem IV.3 for this argument: we impose low ranks ourselves here.

**Lemma VI.2.** *For any $\varepsilon > 0$, there exists an incremental algorithm that maintains $(1-\varepsilon)$-approximate maximum bipartite matching in $O(\varepsilon^{-1}m)$ total time.*

We use a similar idea in order to provide a simple analysis for the offline version of our algorithm. We apply the approximation algorithm with $2/\varepsilon = k = \sqrt{n}$. The running time of the approximation algorithm is, due to Lemma VI.2, $O(m\sqrt{n})$. We are left with at most $2|M^*|/k \in O(\sqrt{n})$ unmatched vertices, so we can afford to perform the full search for the augmenting paths. However, this simple analysis does not carry over if we need to match every black vertex at the moment it is presented.

### B. Decremental Algorithm

Let us consider the online problem where at the beginning we are given a graph $G^d = \langle W^d \uplus B^d, E^d \rangle$, and the vertices from $W$ are removed from $G^d$ in an online manner. During this process we are asked to maintain the maximum size matching in $G^d$. More formally we consider graphs $G_t^d = (W_0^d \uplus B^d, E_t^d)$ constructed by the following scenario:

- we start with $G_0^d = G^d$,
- in turn $t$ we remove vertex $w_t \in W$ along with all its adjacent edges:
$$\begin{cases} E_t^d = E_{t-1}^d \setminus \Big\{\{w_t, b\} : b \in B^d\Big\}, \\ W_t^d = W_{t-1}^d \setminus \{w_t\}. \end{cases}$$
- we finish when $t = |W_0^d|$, i.e., when all white vertices are removed.

Observe the important difference with respect to the incremental problem. Here, we are removing white vertices, whereas previously we were adding black vertices. This change in colors will be useful when reducing the decremental problem to the incremental one. Now we initialize the incremental algorithm with graph $G^d$, i.e., we set $G_0 = G^d$. When the vertex $w_t$ should be removed from $G_{t-1}^d$ we add a vertex $b_t$ to $G_{t-1}$ using the incremental algorithm. The only edge adjacent to $b_t$ that we add to $G_{t-1}$ is $b_t w_t$. Let $G_t$ denote the sequence of graphs resulting from this incremental process. We observe the following:

**Lemma VI.3.** *Let $M_t$ be the maximum size matching in $G_t$, and let $M_t^d$ be the maximum size matching in $G_t^d$. Then $|M_t| - t = |M_t^d|$.*

*Proof.* Consider the matching $M_t' = M_t^d \cup \{b_i w_i : 1 \leqslant i \leqslant t\}$. By this construction we have $|M_t'| - t = |M_t^d|$. If $M_t'$ is not a maximum size matching in $G_t$ then there exists an augmenting path $\pi$ with respect to $M_t'$. This path cannot pass via any of the edges in $\{b_i w_i : 1 \leqslant i \leqslant t\}$, because $b_i$ has degree 1. Hence, $\pi$ is an augmenting path for $M_t^d$ in $G_t^d$ what contradicts the maximality of $M_t^d$. Hence, $|M_t| = |M_t'| = |M_t^d| + t$. □

Combining these results with the algorithm of Section V we obtain the following observation.

**Corollary VI.4.** There exists a decremental algorithm for the maximum bipartite matching problem that works in $O(\sqrt{n}m)$ total time.

We will now prove that the same reduction works in the approximate case as well. In the incremental case we were using Claim VI.1. This fact cannot be used directly with the above reduction, because the sizes of the two matchings $M_t$ and $M_t^d$ differ by $t$. However, it is possible to prove the guarantee on approximation directly form the nonexistence of short augmenting paths.

**Lemma VI.5.** *Let $M_t$ be a matching in $G_t$ such the shortest augmenting path with respect to $M_t$ has length $k$, for some $k \geqslant 1$, and let $M_t^d$ be the maximum size matching in $G_t^d$. Then $|M_t| - t \geqslant \left(1 - \frac{2}{k-2}\right)|M_t^d|$.*

By using the above lemma together with Lemma VI.2 we obtain the following observation.

**Corollary VI.6.** For any $\varepsilon > 0$, there exists a decremental algorithm that maintains $(1-\varepsilon)$-approximate maximum bipartite matching in $O(\varepsilon^{-1}m)$ total time.

### C. Weighted Algorithm

In this section we are going to recall the unfolded graph technique that was introduced in [13]. Let $G_w = (V_w, E_w)$ be a graph where edge weights are given by a function $w : E \to [1, W]$ for some natural number $W$. The *unfolded graph* $G = (V, E)$ of $G_w$ is defined as follows:

$$V = \{v_i : v \in V \text{ and } i \in [1, W]\},$$
$$E = \Big\{(u_i v_{W-i+1}) : uv \in E \text{ and } i \in [1, W]\Big\}.$$

We have the following lemma.

**Lemma VI.7** (Lemma 4.1 of [13])**.** *The weight of the maximum weight matching in $G_w$ is equal to the size of the maximum matching in $G$.*

Using the above lemma we can compute the weight of the maximum matching in $G_w$ online by maintaining the unfolded graph $G$. The unfolded graph has $Wm$ edges and $Wn$ vertices. Hence, the following corollaries are immediate.

**Corollary VI.8.** There exist incremental and decremental algorithms that maintain the weight of the maximum weight matching in $G_w$ that work in $O(W^{3/2}\sqrt{n}m)$ total time.

**Corollary VI.9.** There exist incremental and decremental algorithms that maintain $(1-\varepsilon)$-approximation of the weight of the maximum weight matching in $G_w$ that work in $O(W\varepsilon^{-1}m)$ total time.

## VII. Conclusions

In this paper we have introduced a new greedy framework for solving online maximum matching problem in bipartite graphs. We believe that the introduction of this framework will inspire further work on development of efficient online algorithms for other combinatorial problems like: maximum-flow, min-cost maximum-flow, stable matchings, matroid intersection or non-bipartite matchings. In particular, we would like to stress the following open problems.

1) Is it possible to extend our results to the online non-bipartite matching problem, where vertices of the graph are revealed online? So far only $O(nm)$ time exact solution is known.
2) Can our framework be adopted to the incremental max-flow problem? Here, even in the case where all edge capacities are equal to one, no dynamic solution is known for general graph. We know only some dynamic algorithms in the case of planar graphs [11].
3) We note that our model is different from the edge update scenarios studied in [12], [20], [18], [4], [17], [9], [2], and it would be interesting to see whether our techniques can be applied in this model to get fast incremental or decremantal algorithms?
4) As noted in the introduction the lower bounds developed in [1] do not apply to our model, as they talk about edge updates. Can one show similar lower bounds for vertex-updates?

## References

[1] A. Abboud and V. Vassilevska Williams, *Popular conjectures imply strong lower bounds for dynamic problems*, ArXiv e-prints, (2014).

[2] A. Anand, S. Baswana, M. Gupta, and S. Sen, *Maintaining Approximate Maximum Weighted Matching in Fully Dynamic Graphs*, in IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012), D. D'Souza, T. Kavitha, and J. Radhakrishnan, eds., vol. 18 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2012, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 257–266.

[3] Y. Azar, *On-line load balancing*, in Theoretical Computer Science, Springer, 1992, pp. 218–225.

[4] S. Baswana, M. Gupta, and S. Sen, *Fully dynamic maximal matching in o (log n) update time*, in Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science, FOCS '11, Washington, DC, USA, 2011, IEEE Computer Society, pp. 383–392.

[5] T. M. Chan, *Dynamic subgraph connectivity with geometric applications*, in Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing, STOC '02, New York, NY, USA, 2002, ACM, pp. 7–13.

[6] T. M. Chan, M. Pătraşcu, and L. Roditty, *Dynamic connectivity: Connecting to networks and geometry*, SIAM Journal on Computing, 40 (2011), pp. 333–349. See also FOCS'08, arXiv:0808.1128.

[7] K. Chaudhuri, C. Daskalakis, R. D. Kleinberg, and H. Lin, *Online bipartite perfect matching with augmentations*, in INFOCOM'09, 2009, pp. 1044–1052.

[8] E. Grove, M.-Y. Kao, P. Krishnan, and J. Vitter, *Online perfect matching and mobile computing*, in Algorithms and Data Structures, S. Akl, F. Dehne, J.-R. Sack, and N. Santoro, eds., vol. 955 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1995, pp. 194–205.

[9] M. Gupta and R. Peng, *Fully dynamic (1+ e)-approximate matchings*, 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, 0 (2013), pp. 548–557.

[10] J. E. Hopcroft and R. M. Karp, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM Journal on Computing, 2 (1973), pp. 225–231.

[11] G. F. Italiano, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen, *Improved algorithms for min cut and max flow in undirected planar graphs*, in Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing, STOC '11, New York, NY, USA, 2011, ACM, pp. 313–322.

[12] Z. Ivković and E. Lloyd, *Fully dynamic maintenance of vertex cover*, in Graph-Theoretic Concepts in Computer Science, J. Leeuwen, ed., vol. 790 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1994, pp. 99–111.

[13] M.-Y. Kao, T. W. Lam, W.-K. Sung, and H.-F. Ting, *A decomposition theorem for maximum weight bipartite matchings with applications to evolutionary trees*, in Proceedings of the 7th Annual European Symposium on Algorithms, 1999, pp. 438–449.

[14] R. M. Karp, U. V. Vazirani, and V. V. Vazirani, *An optimal algorithm for on-line bipartite matching*, in STOC, H. Ortiz, ed., ACM, 1990, pp. 352–358.

[15] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani, *Adwords and generalized on-line matching*, in Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on, Oct 2005, pp. 264–273.

[16] V. Mirrokni. personal communication.

[17] O. Neiman and S. Solomon, *Simple deterministic algorithms for fully dynamic maximal matching*, in Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing, STOC '13, New York, NY, USA, 2013, ACM, pp. 745–754.

[18] K. Onak and R. Rubinfeld, *Property testing*, Springer-Verlag, Berlin, Heidelberg, 2010, ch. Dynamic Approximate Vertex Cover and Maximum Matching, pp. 341–345.

[19] M. Patrascu, *Towards polynomial lower bounds for dynamic problems*, in Proceedings of the Forty-second ACM Symposium on Theory of Computing, STOC '10, New York, NY, USA, 2010, ACM, pp. 603–610.

[20] P. Sankowski, *Faster dynamic matchings and vertex connectivity*, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07, Philadelphia, PA, USA, 2007, Society for Industrial and Applied Mathematics, pp. 118–126.