University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Dariusz Leniowski

On Maintaining Online Bipartite Matchings with Augmentations

PhD dissertation

Supervisor

dr hab. Piotr Sankowski

Institute of Informatics
University of Warsaw

September 2015

Author's declaration:
aware of legal responsibility I hereby declare that I have written this dissertation myself and all the contents of the dissertation have been obtained by legal means.

September 28, 2015	.....
date	*Dariusz Leniowski*

Supervisor's declaration:
the dissertation is ready to be reviewed

September 28, 2015	.....
date	*dr hab. Piotr Sankowski*

Abstract

In this thesis we study the setting of online bipartite matchings with augmentations, in which a bipartite graph $G = \langle U \uplus V, E \rangle$ is revealed online in a one-sided vertex-incremental fashion. In other words, the set U is known from the start, while the vertices of V arrive one by one, each together with all its incident edges. Our goal is to maintain the maximum cardinality matching, in particular, the previous decisions are revocable. Nevertheless, although we are allowed to reassign already matched pairs to accommodate the newly arrived vertices, as far as it is possible, we would like to keep the number of such operations low. We approach this problem using two greedy strategies, both based on the classical augmenting paths technique.

First, we investigate the shortest augmenting path algorithm, which each turn augments the current matching by using shortest augmenting paths. It was conjectured that the total length of all such paths is $\mathcal{O}(n \log n)$, but no better bound than the naïve $\mathcal{O}(n^2)$ is known even for trees. In this setting we prove an $\mathcal{O}(n \log^2 n)$ upper bound when the underlying graph G is a tree.

For the case of general bipartite graphs we propose another greedy strategy that tries to minimize the maximum number of times each vertex of U is used by augmenting paths so far. This approach yields a new algorithm that produces augmenting paths that reassign any vertex of U at most $\mathcal{O}(n^{1/2})$ times, implying an upper bound of $\mathcal{O}(n^{3/2})$ on the total length of augmenting paths. We show how to efficiently compute these paths and describe an algorithm that maintains the maximum cardinality matching in $\mathcal{O}(m \cdot n^{1/2})$ total time and a $(1 - \varepsilon)$-approximation algorithm that works in $\mathcal{O}(m \cdot \varepsilon^{-1})$ time. Moreover, we extend these results to the decremental case where we give the same total bound. Furthermore, we obtain a pseudo-polynomial algorithm for weighted graphs. Finally, we provide examples demonstrating that our analysis is tight.

Keywords: *online matchings, bipartite matchings, approximate matchings, shortest augmenting paths, dynamic graph algorithms.*
AMS Classification: 05C70, 05C85, 68Q25, 68W25, 68W27, 68W40.

Streszczenie

Tematem pracy jest utrzymywanie skojarzeń online w grafach dwudzielnych za pomocą ścieżek powiększających. W rozważanym problemie algorytm otrzymuje na wejściu graf dwudzielny $G = \langle U \uplus V, E \rangle$, w ten sposób, że zbiór wierzchołków U jest znany od początku działania algorytmu, podczas gdy wierzchołki z V są odkrywane pojedynczo, każdy ze wszystkimi incydentnymi do niego krawędziami. Celem jest utrzymywanie najliczniejszego skojarzenia w ujawnionym grafie. Decyzje algorytmu nie są nieodwołalne, niemniej jednak, na ile to możliwe, dąży się do zminimalizowania ilości zmian. W pracy tej analizowane są dwie zachłanne strategie, obydwie oparte o klasyczną technikę ścieżek powiększających.

Pierwsze podejście opiera się na algorytmie, który w każdej turze aktualizuje bieżące skojarzenie używając najkrótszej z dostępnych ścieżek powiększających. Według hipotezy, suma długości wszystkich takich ścieżek jest ograniczona przez $\mathcal{O}(n \log n)$. Tymczasem, nawet dla drzew nie znamy żadnego ograniczenia lepszego niż $\mathcal{O}(n^2)$. Niniejsza praca przedstawia dowód, że jeżeli graf G jest drzewem, to całkowita ich długość nie przekracza $\mathcal{O}(n \log^2 n)$.

Dla przypadku ogólnych grafów dwudzielnych rozważana jest strategia, która zlicza ile razy każdy wierzchołek był używany przez dotychczasowe ścieżki powiększające i stara się zachłannie zminimalizować maksimum. Zaproponowane podejście owocuje nowym algorytmem, w którym każdy wierzchołek z U jest użyty przez ścieżki powiększające co najwyżej $\mathcal{O}(n^{1/2})$ razy, a całkowita suma długości tych ścieżek jest ograniczona przez $\mathcal{O}(n^{3/2})$. Ponadto, w pracy zaprezentowany jest efektywny sposób ich obliczania, który pozwala na uzyskanie procedury utrzymującej najliczniejsze skojarzenie w całkowitym czasie $\mathcal{O}(m \cdot n^{1/2})$ oraz algorytmu $(1-\varepsilon)$-aproksymacyjnego działającego w czasie $\mathcal{O}(m \cdot \varepsilon^{-1})$. Wyniki te zostają rozszerzone do przypadku dekrementalnego i algorytmu pseudowielomianowego dla grafów ważonych. Na koniec podane są przykłady, które pokazują, że zastosowana analiza czasowa jest ścisła.

Słowa kluczowe: *skojarzenia online, skojarzenia dwudzielne, skojarzenia aproksymowane, najkrótsze ścieżki powiększające, dynamiczne algorytmy grafowe.*
Klasyfikacja AMS: 05C70, 05C85, 68Q25, 68W25, 68W27, 68W40.

Acknowledgements

First and foremost I deeply thank Piotr Sankowski, my supervisor. For reasons unknown you had a lot of trust in me. I am grateful that you would treat me like your equal, and that you were a boss when I needed it. Needless to say, without your guidance this work would not have come to fruition. Despite your unpretentious nature and self-effacing ways, you have an extraordinary influence upon those around you, and I am very glad that I could have been a part of that group.

Special thanks must be made to Anna Zych, for this thesis would not be completed without her help and advice. It is hard to overstate how much I appreciate all these long hours you spent patiently listening to me spewing gibberish, and the insightful remarks that each time allowed me to move forward. We had our disagreements, yet, for them I am especially grateful, as it was at those moments, when I grew as a researcher.

I wholeheartedly thank Bartłomiej Bosek, with whom the original problem and the concept of ranks were conceived. At that time I was just some exchange visitor in Theoretical Computer Science Department of Jagiellonian University, Kraków. The warm welcome I got from you I remember even now. It was your keen and open mind that was able to appreciate the setting we stumbled upon at first sight, and made this whole endeavor possible.

During my graduate studies I had the privilege to meet and learn from many wonderful people. Although it would be impossible to include here the whole list, I would like to express my gratitude for their efforts and help I received. Nonetheless, I cannot go without mentioning by name a few that I am most indebted with. I would like to thank Gianpiero Monaco, with whom I collaborated on my first published conference paper—I have learned a lot from you, and I am glad I got to know you. I wish to acknowledge Aranyak Mehta, Chetan Parampalli and Venu Vemula, whom I had met during my internship with Google Inc. in 2012—it was thanks to you that I got interested in the Adwords problem and online bipartite matchings. Next year, thanks to Stefano Leonardi, I was able to visit Sapienza University of Rome—although our collaborative efforts did not bear fruit, I am grateful for your unprecedented hospitality and all

the things I could learn from you and people in your group. Then, two years after my trip to California, I was invited by the New York team of Google Research; my deepest gratitude goes to Evan Benshetler, Nitish Korula, Renato Paes Leme and Vahab Mirrokni—working with you was a real pleasure, to see the extent to which you have successfully merged theory and practice was a truly revealing experience.

These are the people who, from a professional perspective, had the most influence on this work. Yet, there are still a few more persons whose contributions might not be as obvious and clear, but are no less important. I would like to express my gratitude to Krzysztof Diks—your deep heartfelt passion for algorithms and data structures remains with me until this very day. Special thanks to Jakub Łącki—for all the things you did and for those which you would do, even if I would not have asked. Heartfelt thanks to Łukasz and Maria Bieniasz-Krzywiec—for your help over the years, particularly when I was starting my graduate studies.

I am forever indebted to my family. Especially, to my parents—for your endless unwavering support. Even more so, to my wife Joanna, whose love, trust and sacrifice gave me the strength and will to finish this endeavor—to you and our son Sebastian, a word of apology—I am deeply sorry for the many occasions on which I neglected you while working on this thesis.

Above all, I wholeheartedly thank the mighty God for His gifts that allowed me to pursue this interesting research.

This thesis was completed during the Środowiskowe Studia Doktoranckie z Nauk Matematycznych graduate program. The research was conducted together with Bartłomiej Bosek, Piotr Sankowski and Anna Zych. It was partially supported by ERC StG project PAAl 259515 and FET IP project MULTIPEX 317532. A number of results included in this work were published before. In particular, Theorem 2.6 and the claims it depends on appeared in *"Shortest Augmenting Paths for Online Matchings on Trees"* at WAOA 2015 [19], while the theorems of Chapters 3 and 4 were presented at FOCS 2014 within a paper titled *"Online Bipartite Matching in Offline Time"* [18].

Contents

List of Figures

Introduction

Even before our ancestors became entirely human, they had to allocate resources—how much energy to spend on foraging, how much on other survival necessities, how much time on rest and reproduction? Often, the choice was not even conscious, but encoded in their genes and dictated by the rhythm of the planet. Nowadays, as we live our lives, resource allocation is ubiquitous. As kids we learn to balance our responsibilities and pleasures, as adults we pick roles to fulfill and attempt to find an equilibrium between work, friends and family.

Yet, our pursuit of efficiency takes us even further. Even meta-level considerations are not free of resource allocation, as planning itself also consumes time and energy. Given how our existence is intertwined with technology, it is not surprising that we use science and engineering to help us with resource allocation. In this thesis we focus on an algorithmic approach to one of many such problems. More precisely, its main result is a procedure for maintaining a maximum cardinality matching.

A matching M in a graph $G = \langle V, E \rangle$ is any subset of edges $M \subseteq E$ that are pairwise vertex-disjoint, i.e., each vertex has at most one incident edge in M. For example, consider a group of people interconnected by a symmetric relation of acquaintance. A matching then is a pairing of these people such that each person in a pair is acquainted with the other and nobody is in a more than one pair. We call M a maximum cardinality matching, or simply a maximum matching, if it is of biggest possible size, that is, for any other matching M' we have $|M| \geqslant |M'|$.

It is easiest to see the task of calculating maximum matching as a resource allocation problem in the bipartite case, that is, when the vertices

V of G can be partitioned into two sets such that all the edges connect vertices from two different parts. Despite this limitation, practical applications of bipartite matching are numerous, especially, when each edge is labeled by weight that represents the benefit or cost associated with it. For example, medical students in the United States have been assigned to hospitals using a similar setting since the early 50′ of the last century. Even before, minimum weight matching was used to optimize problems motivated by transportation or classification of military personnel. Furthermore, both weighted and unweighted versions of the maximum matching problem found multiple uses as building blocks of more complex algorithms, having applications in mobile sensing systems, recommendation systems, algorithms including pattern recognition and several areas of bioinformatics, as well as in theoretical research on scheduling and load balancing, shortest paths algorithms and hashing.

Over the years, as the research in combinatorial optimization progressed, the notion of matching became one of the fundamental concepts in graph theory. Today it has many different versions and flavors, generalizations and special cases. The subject of this thesis, that is, the problem of maintaining a maximum cardinality matching in an incremental setting, is one such specific variant, which can be summarized as follows. Consider a number of incoming continuous service requests and a known pool of handlers of different abilities—perhaps teams of people, each group able to take care of one job that falls into its area of expertise, or maybe just a farm of servers, each taking care of one request within its capabilities range. Our goal is to continue to maintain as many requests as possible, at the same time minimizing the number of reassignments between the teams and their jobs or between servers and clients connected to them. As argued in [24], possible applications of this problem include streaming content delivery, data storage, job scheduling and hashing.

That setting is an example from a wide class of problems, in which we assume that the input is revealed gradually as the algorithm runs, rather than the all data being given as a whole from the start. Thanks to the increase of easily available computational power, the online approach became not only interesting from theoretical perspective, but also practi-

cal on a massive scale. For example, these days, when visiting a website that uses one of the big ad content providers, during the time in which the browser loads the page, huge server farms perform multiple small auctions that decide which vendors will be granted ad space in that particular page impression. Nevertheless, even this vast computing power is not enough to deal with ever-growing amount of information to process—the current monthly internet traffic, as well as the size of some larger databases are measured in exabytes. That forces us to consider algorithms of sublinear efficiency, but how is it possible for the algorithm to run without reading all the data? In a dynamic approach we leverage the fact that large data sets often differ very little from turn to turn, i.e., it is possible to make the procedure run faster by reusing some of the calculation it did on previous rounds.

The online and dynamic approaches are two perspectives on procedures that process data arriving during the algorithm run. The difference between the two depends on the context in which we do the analysis. If we focus on the quality of the solution and decisions made by the algorithm, then we would say it is an online problem, while in a dynamic problem we concentrate on the efficiency of handling the updates.

A perfect example of an online problem is the one studied by Karp, Vazirani and Vazirani in [66], a work that has hundreds of citations and spurred a whole line of academic research. The input is a bipartite graph $G = \langle U \uplus V, E \rangle$ where the vertices of V arrive online, each $v \in V$ being revealed at the same time with all its incident edges. When this happens, the algorithm has to make a decision to leave v unmatched or pair it up with some adjacent vertex $u \in U$ which is still free. Once the choice is made, it is irrevocable, i.e., the matched vertices cannot change their pairs and the unmatched vertices of V have to stay unmatched until the end. The objective is to maximize the size of the resulting matching, and the authors propose an algorithm that has $(1 - 1/e)$ competitive ratio, which means that the size of the computed solution is at least $(1 - 1/e)$ times the size of the optimal offline matching. In other words, they analyze the quality of the output by comparing it to the optimal offline solution calculated on the whole final graph G.

On the other hand, in the fully dynamic bipartite matching problem we consider the efficiency of the algorithm per single update. In that setting the input graph G is given as a series of both insertions and deletions of edges intermixed with each other, so it is possible for one edge to be added and removed multiple times. The challenge is to develop an algorithm that maintains the maximum cardinality matching and has the fastest possible update times. The online approach does not apply, because each time we calculate the best solution available. Conversely, the dynamic approach does not suit the previous problem, because we are not concerned with efficiency, only how good the final solution will be.

Nevertheless, the distinction between online and dynamic algorithms is not sharply defined. For example, the problem studied in this thesis has characteristics of both—it can be thought of as a middle ground between the two above settings. In a graph-theoretical language, we can formulate it as follows. We are given a bipartite graph $G = \langle W \uplus B, E \rangle$ in a one-sided online fashion, that is, each black vertex $b \in B$ arrives together with all its incident edges, exactly as in the work of Karp, Vazirani and Vazirani. However, similarly to the dynamic problem, our goal is to maintain a maximum cardinality matching. In particular, the decisions of the algorithm can be changed later—each time a black vertex $b \in B$ is revealed, the algorithm picks its white pair in W, potentially reassigning some other vertices in the process. The problem can be considered dynamic, because we try to calculate, in an efficient manner, an optimal matching for each turn. Even so, in a resemblance to the online setting, we still want to keep the number of rematchings low. In other words, the fewer reallocations are necessary, the better the quality of our solution.

Matching problems are far from being the only problems in the online and dynamic settings. Still, the steadily growing body of research on this topic and the number of papers accepted by the best algorithmic conferences, like IEEE's Foundations of Computer Science [76, 41, 10, 49, 75, 84, 55] and ACM's Symposium on Theory of Computing [66, 81, 65, 71, 80], is an evidence of its importance for the algorithmic graph theory. In that light it is not surprising that the already huge volume of work and the pace of new developments make it impossible to do an extensive survey

within scope of this thesis. Nonetheless, to make the context more available, in the remainder of this chapter we try to sketch some recent and the most significant advances in the area. The introduction ends with an informal, but detailed description of the obtained results, including the links to chapters in which they can be found.

Related Work

The exact setting we are working with, namely the problem of online bipartite matching with augmentations, was introduced by Grove, Kao, Krishnan and Vitter in [53]. The authors used competitive analysis to show $\mathcal{O}(\log|V|)$ bound when each client connects to at most two servers.

That result was then extended by Chaudhuri, Daskalakis, Kleinberg and Lin, who proved $\mathcal{O}(|V|\log|V|)$ bound for some restricted models, including forests or random graphs with degree $\Theta(\log|V|)$. They also showed $\mathcal{O}(|V|\log|V|)$ bound with high probability for the shortest augmenting path algorithm when the clients arrive in random order.

More recently Gupta, Kumar and Stein [54] considered a similar setting that allows the capacity constrains to be exceeded by a constant factor. They designed an algorithm that maintains such a constant-factor load using only a constant number of reassignments per vertex in the amortized sense.

Unfortunately, these are the only works that consider a model that is the same or really close to ours. Still, there are three general lines of research that are related to the topic of this thesis: online matching algorithms, dynamic matching algorithms and load balancing problems. Nevertheless, to make the context complete, before we review the previous results in these areas, we start with the matching algorithms in the offline setting.

Offline Matching The matching theory has a long history [69, 31], with published papers dating back to the nineteenth century [83]. Only recently it was discovered that at that time Carl Gustav Jacobi studied what we would call today—with a bit of a stretch, because the formulation, lan-

guage and methods differ greatly—the offline assignment problem [22]. As the resource allocation problems are practical, their applications followed [58, 63, 91] .

The classic approach to matching problems is via augmenting paths. Thanks to a lemma proven by Berge (see Theorem 1.3) [12], we know that in the case of bipartite graph the maximum cardinality matching problem can be solved by the trivial algorithm that repeatedly applies arbitrary augmenting paths—it works in $\mathcal{O}(|V| \cdot |E|)$ time.

The non-bipartite case was later solved by Edmonds [34] in $\mathcal{O}(|V| \cdot |E|^2)$ and then improved to $\mathcal{O}(|V| \cdot |E|)$ by a number of authors [31].

Another progress was made by Hopcroft and Karp [59] who proposed an $\mathcal{O}(|E| \cdot |V|^{1/2})$ algorithm for the bipartite case, that was later improved to handle the general case by Mical and Vazirani [78]. It is worth noting that the running time of the latter algorithm depends on a special disjoint union structure developed later by Gabov and Tarian [46], and without it the actual running time is $\mathcal{O}(|E| \cdot |V|^{1/2} \cdot \alpha(|E|, |V|))$ [92].

Then, Alt, Blum, Melhorn and Paul [4], and Feder and Motwani [38] showed logarithmic improvements, that is, $\mathcal{O}(|E|^{1/2} \cdot |V|^{3/2} \cdot \lambda^{-1})$ for $\lambda = \log^{1/2} |V|$ and $\mathcal{O}(|E| \cdot |V|^{1/2} \cdot \kappa^{-1})$ for $\kappa = \frac{\log |V|}{\log |V|^2 |E|^{-1}}$ espectively. While both results apply only to the bipartite case, the latter presents a graph compression technique that was later applied by Goldberg and Kennedy [51] to the push-relabel algorithm, and allowed Goldberg and Karzanov [50] to design an algorithm for the general graphs with the same running time.

In the meantime, there was a breakthrough by Ibarra and Moran [60], who used randomization and algebraic techniques to calculate the cardinality of the maximum matching in the time of matrix multiplication $\mathcal{O}(|V|^{\omega})$. Later, Rabin and Vazirani [85] improved that result to handle general graphs in the same time and calculate the whole matching in $\mathcal{O}(|V|^{1+\omega})$. Lastly, Mucha and Sankowski [79] managed to reduce the computation of the matching to $\mathcal{O}(|V|^{\omega})$ time, and thus obtained an algorithm which remains the fastest in the case of dense graphs.

With such a long history, it might appear that the problems of general matching and bipartite matching are hard to improve. Nevertheless, at FOCS 2013, Aleksander Mądry [70] presented a design of $\tilde{\mathcal{O}}(|E|^{10/7})$-time

algorithm for the maximum cardinality bipartite matching, which is a significant advancement for sparse graphs. Moreover, in another recent work, Duan and Pettie [31] had proven $(1-\varepsilon)$-approximation for their maximum weight matching procedure that works in $\mathcal{O}(|E| \cdot \varepsilon^{-1} \log \varepsilon^{-1})$ time. Finally, it is worth to mention that their paper contains also an excellent section on related work to which we direct the reader for more details on the maximum matching problems.

Online Problems The problem of online bipartite matching, where vertices from one side arrive online together with all their edges, was introduced in a paper by Karp, Vazirani and Vazirani [66]. The authors apply the framework of competitive analysis and show an algorithm that has $(1-1/e)$ approximation factor, a result that was later greatly simplified by Birnbaum and Mathieu [16].

Although the model might seem restricted, actually it has numerous applications, mainly because it is useful to model two sided markets—at the same time it captures some non-trivial characteristics, and yet is simple enough to allow an in-depth analysis. While it is rare that we know both the supply and the demand, often it is not unreasonable to assume that one of them can be reasonably assessed. For example, it might be hard for a hosting company to accurately predict the incoming daily traffic, but it can easily estimate the load that its servers can withstand.

Thus, in 2005 Metha, Saberi, Vazirani and Vazirani [76] generalized the online bipartite matching to accommodate online advertising and present worst-case $(1-1/e)$-approximation for this special setting, when budgets of advertisers are large. Another algorithm with same approximation factor was given by Buchbinder, Jain and Naor [21], who use the primal-dual approach to obtain their result.

Later, Goel and Mehta [48] went back to analyse the basic greedy algorithm to show it is also $(1-1/e)$-competitive, if the input is presented as independent samples from some known distribution (the so-called *i.i.d.* model), or it arrives in random order. In the case of the random permutation model they give lower bounds for the approximation factors of any deterministic and randomized algorithm, which are respectively $3/4$ and $5/6$.

The natural barrier of $(1 - 1/e) \approx 0.63$ was "beaten" by Feldman, Mehta, Mirrokni and Muthukrishnan [41] in the case of the i.i.d. model, and by Devenur and Hayes [29] in the random permutation model under several additional assumptions. Then, with a different set of assumptions, Feldman, Henzinger, Korula, Mirrokni and Stein [39] obtained $(1 - \varepsilon)$-approximation algorithm motivated by a practical approach of training on past historical data. After that, a number of papers improved both upper bounds and lower bounds, finally arriving at 0.696-approximation in the random arrival model and 0.702-approximation together with 0.823 lower bound in the case of known distribution [72, 65, 71]. A good overview of these results was given in a survey by Mehta [74].

A variety of other related models were studied as well. The most straightforward generalization involved adding weights. For example, Aggrawal, Goel, Karande and Mehta [2] achieved $(1 - 1/e)$-competitive algorithm in the adversarial vertex-weighted case, while weights on edges were considered by Haeupler, Mirrokni and Zadimoghaddam [56], who presented an algorithm that achieves 0.667-approximation when the input distribution is known.

Then, in 2012 the audience of FOCS have seen three interesting results. Poloczek and Szegedy [84] investigated a middle ground between online and randomized offline settings in which they show that the approximation factor of the greedy algorithm is strictly better than $1/2$. On the other hand, Goel and Tripathi analyzed a case where the graph is revealed as it is accessed, as if the algorithm had its "eyes closed". They presented a 0.56 upper bound for their greedy algorithm and 0.7916 lower bound if the procedure always matches just discovered edges. The setting of the third work was inspired by practical application, namely the problem of user conversion in online advertising. Mehta and Panigrahni [75] incorporated an additional probability parameter p that indicates how likely the allocation is successful, and gave a deterministic algorithm with its competitive ratio converging to 0.567. Just this year, with a small drop to 0.534-approximation, their result was generalized by Mehta, Waggoner and Zadimoghaddam [77] to probabilities that may be unequal between vertices.

Another setting motivated by practice was the bicriteria online matching in the work of Korula, Mirrokni and Zadimoghaddam [67]. To model capacity constraints found in online advertising, they endowed the static vertices with numbers and ensure that they are matched at most that many times. The authors presented a parametrized approximation algorithm which balances the tradeoff between the cardinality constraints and maximizing the weight.

A bit closer to our setting is the *free disposal* model introduced by Feldman, Korula, Mirrokni, Muthukrishnan and Pál in [40]—the vertices are allowed to be matched more than once, but they can get credit only for a few of its matched edges. They show $(1 - 1/k)^k$-approximation algorithm if all the capacities exceed weakly k. The hard case of unit capacities was recently considered by Charikar, Henzinger and Nguyen [23] who designed a 0.5664-competitive algorithm if the graph is complete and the weights of the edges are products of numbers associated with the endpoints of that edge.

Similar to the free disposal model is the problem proposed by Constantin, Feldman, Muthukrishnan and Pál [26]—reservation with cancellations, where the matched edges may be revoked at a cost. The same problem was also studied by Babaioff, Hartline and Kleinberg [9], and Ashwinkumar and Kleinberg [6].

It is also worth mentioning the streaming model in which the elements arriving online are the edges, and the challenge is to calculate a matching in restricted memory. For example McGregor [73] described an algorithm that achieve $(1/1+\varepsilon)$ and $(1/2+\varepsilon)$ approximation factors for the maximum cardinality matching and maximal weight matching in $\tilde{O}(|V|)$ space and constant number of passes. Epstein, Levin, Segev and Weimann [36] in their work also considered edge-based input, but they allowed for preemptions—a model similar to the one with free disposal, where a previously matched edge may be discarded. For that setting the authors give an upper bound of 0.590 and a randomized algorithm that gives 0.1867-approximation.

There are many other different online models that fit into the general area of online resource allocation, e.g., the generalized assignment problem. However, given the current pace of research, reviewing every single

setting and its modifications is an impossible task. Hence, we move to the domain of dynamic problems.

Dynamic Problems Similarly to other matching problems, the dynamic matching problem has received, especially recently, a lot of attention [62, 3, 20, 88, 81, 10, 5, 55, 80, 15, 14, 11]. However, it is important to note, that the fully-dynamic model is different than the one analyzed in this thesis. Not only it considers edge updates, but it allows their insertions and deletions to be mixed.

Ivkovic and Lloyd [62] studied the problem of dynamic vertex cover, but their solution is based on an algorithm for a maximal matching that works in $\mathcal{O}\big((|V|+|E|)^{\sqrt{2}/2}\big)$ time. Two years later Alberts and Henzinger [3] gave an algorithm that achieves $\mathcal{O}(|V|)$ amortized time per update if only insertions of edges are allowed, and an algorithm with average $\mathcal{O}(|V|)$ time per update in case of the *restricted randomness* model.

Next, in 2007 Brodal, Georgiadis, Arnsfelt and Katriel [20] constructed a data structure that works for *convex* bipartite graphs and has an amortized update time of $\mathcal{O}(\log^2|V|)$. It allows for worst-case constant-time queries whether a given vertex is matched, or $\mathcal{O}\big(\min\{\#\text{updates}\cdot\log^2|V|+\log|V|,\ |V|\log|V|\}\big)$ to find its matched pair.

The same year Sankowski [88] used algebraic techniques to develop an algorithm for maximum matching size with $\mathcal{O}(|V|^{1.495})$ update time and extends it to maximum bipartite matching weight via the unfolded graph technique of [64], achieving update time of $\mathcal{O}(W^{2.495}\cdot|V|^{1.495})$.

Then, at STOC'10 Onak and Rubinfeld [81] presented a constant approximation algorithm that works in polilogarithmic time, a work that was a year later superseded by Baswana, Gupta and Sen [10, 11], who obtained $\mathcal{O}(\log|V|)$ update time and factor two approximation. That result was then extended by Anand, Baswana, Gupta and Sen [5] to weighted matchings, that is, they achieved $1/8$-approximation in expected amortized $\mathcal{O}(\log|V|\cdot\log c)$ time, where c is the bound on the ratio of weights of edges.

In 2013 there were two results with essentially same running time: papers by Neiman and Solomon [80] from STOC and Gupta and Peng [55]

from FOCS. The former provides $3/2$-approximation in $\mathcal{O}(|E|^{1/2})$, while the latter achieves $(1+\varepsilon)$-approximation in $\mathcal{O}(|E|^{1/2} \cdot \varepsilon^{-2})$ time.

Recently, the $\mathcal{O}(|E|^{1/2})$ barrier for the deterministic algorithms was broken by Bhattacharya, Henzinger and Italiano [15]. The authors are able to obtain $(3+\varepsilon)$-approximation in $\mathcal{O}(\min\{|V|^{1/2} \cdot \varepsilon^{-1}, |E|^{1/3} \cdot \varepsilon^{-2}\})$ amortized time per update, and $(4+\varepsilon)$-approximation in $\mathcal{O}(|E|^{1/3} \cdot \varepsilon^{-2})$ worst-case time per update. The same year Bernstein and Stein [14] improved the approximation factor to $(3/2+\varepsilon)$, with worst-case update time of $\mathcal{O}(|E|^{1/4} \cdot \varepsilon^{-2.5})$.

In the last few years there was also some progress in relation to lower bounds on the running time of dynamic algorithms, for example by Patrascu [82] and Abboud and Vassilevska Williams [1]. The second of these papers, with certain assumptions, shows polynomial lower bounds on the running time of incremental and decremental maximum matching algorithms. However, due to edge-based updates, this result is incompatible with our setting. Intuitively, the problem is that a reduction in the required direction would imply adding vertices to both sides of the bipartite graph. On the other hand, we heavily rely on the fact, that only one side of the graph is revealed and each vertex comes together with all its edges. Therefore, it is hard to apply this results to our model.

Scheduling and Load Balancing Finally, we note that our model is related to the area of scheduling and load balancing. One can view the setting as a number of servers waiting for tasks to be assigned—when a new task arrives, before we schedule it for being run on some machine, we might need to migrate some jobs to other servers to accommodate the load. The closest problems are related to online load balancing of permanent tasks with preemption.

As the domain of load balancing has been studied for a long time, for the results published before 1996 we refer to the survey of Azar [8].

More recently Aspnes, Azar, Fiat, Plotkin, and Waarts [7] considered a setting resembling online dynamic matching—online virtual circuits which are permanent, that is, once the circuit has been created it will not be rerouted. They proved $\mathcal{O}(\log |V|)$ congestion factor and presented

$\mathcal{O}(\log |V|)$-competitive algorithm for the non-preemptive load balancing of unrelated machines.

For related machines, where a job of size p requires time p/v on a machine with speed v, Berman, Charikar and Karpinski [13] obtained an algorithm with the approximation factors of 0.171 and 0.232 for the deterministic and randomized versions respectively. The authors also provide an upper bound of 0.5443, which is later improved to $1/2$ in the case of both preemptive and non-preemptive online scheduling by Epstein and Sgall [37].

Load balancing models in which not every task can be assigned to every server are called *restricted assignment* problems. Such a model, among others, was considered by Westbrook [93]. He gave a rebalancing scheme with a constant competitive ratio at a restart cost of $\mathcal{O}(\sum_{v \in V} r_v \log |V|)$, where r_i is the cost of reassignment of i'th job.

Yet another way of modeling the cost of reassignment was investigated by Sanders, Sivadasan and Skutella [87]. Their bounded migration setting constraints the total size of moved jobs by a migration factor β times the size of the arriving job. The authors developed $3/2$-approximate algorithm for $\beta = 4/3$ and general $(1+\varepsilon)$-approximation for a constant $\beta(\varepsilon)$ dependent on ε. The migration factor also considered by Epstein and Levin in [35]. They presented an algorithm of factor $\beta = 1 - 1/|E|$ maintaining a solution on identical machines which is optimal with respect to the makespan minimization and any ℓ_p norm for $p > 1$.

Nevertheless, as similar as these models might seem, the results are not quite applicable to our setting. This is because 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.

Results

In this thesis we consider the setting of online bipartite matchings with augmentations, introduced by Grove, Kao, Krishnan and Vitter in [53]. Shortly, we are given a bipartite graph $G = \langle W \uplus B, E \rangle$ with its vertices partitioned into static white set W and black set B, that is revealed online in a one-sided vertex-incremental fashion. In other words, the set of white vertices W is fixed and assumed to be known beforehand, while the vertices of B arrive one per turn, each with all its incident edges. Our aim is to *maintain* the maximum cardinality matching, i.e., to be able to output at any given time an optimal matching in a graph that was revealed up to this point. Also, as far as it is possible, we would like to keep the number of reassignments of the vertices low.

We approach this problem using two greedy strategies, both based on the classical augmenting paths technique. The first one is presented in Chapter 2, while the other spans Chapters 3 to 5. A formalized setting description is given in Section 1.2.2 of the preliminaries, Chapter 1. The final chapter of the whole thesis contains a brief discussion on the obtained results and potential interesting avenues for further research.

A number of results included in thesis were published before, in particular Theorem 2.6 appeared in "Shortest Augmenting Paths for Online Matchings on Trees" at WAOA'15 [19], while Theorem 3.16 and Algorithm 3 appeared in "Online Bipartite Matching in Offline Time" at FOCS'14 [18]. It is worth mentioning that prior to these papers the best bounds known for the general case were $\mathcal{O}(|V|^2)$ and $\mathcal{O}(|E| \cdot |V|)$, respectively, for the total length of augmenting paths and the running time, which corresponds to the naïve algorithm that repeatedly applies just any augmenting path.

Shortest Paths Approach The first strategy, described in Chapter 2, uses the shortest augmenting paths. Chaudhuri, Daskalakis, Kleinberg and Lin [24] conjecture that such a method achieves $\mathcal{O}(|V| \log |V|)$ upper bound on the total number of assignments and reassignments. To this end, in Section 2.1 we present a reduction that may potentially simplify any dynamic

analysis—the original graph G is transformed into a problem instance in which only vertices of degree 1 are added. This way, the whole structure of G is static and available from the very first turn.

Then, in Section 2.2, we consider the special case when G is a tree and prove the bound of $\mathcal{O}(|V| \log^2 |V|)$ on the total length of augmenting paths. That proof is special, because it does not rely on the actual matching it augments each turn, only on the structure of the whole graph. This was possible, thanks to how the concept of *minimum surplus* was defined in Section 1.2.2—although ideas based on Hall's surplus are known and appear in literature in various forms, the author has never seen the formulation used in Definition 1.10.

Therefore, in Section 2.3 we propose a setting in which the adversary can change the calculated matching at every turn, in any way that preserves its cardinality. We conjecture that this intervention does not affect the asymptotic length of all the augmenting paths produced by the shortest augmenting paths algorithm.

Ranks and Tiers The second strategy we apply to the online bipartite matching with augmentations is based on another heuristic. For each vertex we keep count of how many times it was used by augmenting paths, which we call a *rank*, and try to minimize the maximum of these values over all the vertices. Chapter 3 provides definitions and the theoretical foundations of the approach. Then, we use them in Chapter 4 to design an efficient algorithm that maintains a maximum cardinality bipartite matching in the online setting with augmentations. Chapter 5 describes a number of examples that demonstrate the performance of the technique and provide corresponding lower bounds on running time of the algorithms and the total length of applied augmenting paths.

The two most important notions of the second strategy are the rank mentioned above and a related concept of tier, both defined in Section 3.1. Section 3.2 provides their most important properties, while Section 3.3 contains the main theoretical result of this thesis, namely Theorem 3.16, which can be formulated as presented below. Finally, in Section 3.5 we relax the definitions a bit, so that they are easier to use in an algorithmic

setting. In particular, Theorem 3.36 restates Theorem 3.16 using the new relaxed rank.

Theorem 3.16. *For any dynamic unweighted matching algorithm that uses tiered augmenting paths it holds that* $\operatorname{rank}_{|B|}(w) \leqslant \left(2|W \uplus B|\right)^{1/2}$ *for every* $w \in W$.

The above theorem allows us to design a number of algorithms for the online bipartite matching problem with augmentations and its different flavors, which we all present in Chapter 4. We begin with Algorithm 1, the search procedure which constitutes the heart of all the other algorithms. Section 4.2 describes two results, which are the main practical contribution of this work, namely Algorithm 3 and Algorithm 4.

Both procedures work in $\mathcal{O}\left(\mathcal{R} \cdot |E|\right)$ total time and generate augmenting paths of $\mathcal{O}\left(\mathcal{R} \cdot |V|\right)$ total length, where $\mathcal{R}$ is the maximum achieved rank. The first algorithm maintains an *exact* maximum cardinality matching, due to Theorems 3.16 and 3.36 its worst-case total running time is bounded from above by $\mathcal{O}\left(|E| \cdot |V|^{1/2}\right)$. For the same reason the sum of lengths of augmenting paths it produces is $\mathcal{O}\left(|V|^{3/2}\right)$. The second algorithm is parametrized by a positive $\varepsilon > 0$ and maintains an $(1-\varepsilon)$-approximation of the maximum cardinality matching, that is, a matching M such that for any other matching M' we have $|M| \geqslant (1-\varepsilon)|M'|$. As the procedure keeps the maximum rank $\mathcal{R}$ bounded by $\mathcal{O}\left(\varepsilon^{-1}\right)$, it works in $\mathcal{O}\left(|E| \cdot \varepsilon^{-1}\right)$ total time and generates paths of $\mathcal{O}\left(|V| \cdot \varepsilon^{-1}\right)$ total length.

It is worth pointing out, that both $\mathcal{O}\left(|E| \cdot |V|^{1/2}\right)$ and $\mathcal{O}\left(|E| \cdot \varepsilon^{-1}\right)$ happen to be the same as the asymptotic times needed by the well-known algorithm of Hopcroft and Karp [59] to compute the exact and $(1-\varepsilon)$-approximate maximum cardinality matchings in offline setting. Thus, in Section 4.3 we compare the behavior of our algorithm to other offline matching procedures and formulate a conjecture that Algorithm 8, a randomized version solving the offline maximum bipartite matching problem, has expected running time of $\mathcal{O}\left(|E| \cdot |V|^{\alpha}\right)$ time for some $\alpha < 1/2$.

We finish Chapter 4 with two reductions: from decremental-only case and from weighted case. The first considers a setting in which we are given a bipartite graph G and each turn instead of adding a vertex we remove one. In Section 4.4 we show that Algorithm 3 and Algorithm 4 can be trans-

formed to handle the decremental case in the same running times. Then, in Section 4.5 we use the unfolded graph technique of Kao, Lam, Sung and Ting [64] to apply all the above algorithms and their modifications to weighted graphs. In this way we obtain an incremental and decremental algorithms that maintain the exact weights of the maximum weight bipartite matchings in $\mathcal{O}\left(W^{3/2} \cdot |E| \cdot |V|^{1/2}\right)$ and their $(1-\varepsilon)$-approximations in $\mathcal{O}\left(W \cdot |E| \cdot \varepsilon^{-1}\right)$ total time.

Finally, Chapter 5 presents a number of examples that demonstrate various characteristic behaviors of Algorithm 3 and Algorithm 4 and provide lower bounds on the pessimistic ranks and tiers. We consider two methods of causing high ranks, one that targets the searching procedure, and another that focuses on the augmenting paths.

The former strategy is discussed in Section 5.1, in which we construct for a given parameter r three problem instances that achieve a cubic sum of ranks $\Omega(r^3)$ while being of sizes $\mathcal{O}(r^2)$, $\mathcal{O}(r^2)$ and $\mathcal{O}(r^2 \log r)$ respectively. We start with an exposition of a general idea using a simple example for a version of our matching algorithm that is based on the breadth-first search. Then we proceed to modify it to accommodate Algorithm 3, which is based on the depth-first search. However, the second example works only in the worst-case. To this end we introduce the third example, that achieves an expected $\Omega(r^3)$ sum of ranks even if the edges of any vertex are examined in a random order.

The shortcoming of these examples is that, the total length of applied augmenting paths is linear in $|V|$. Thus, in Section 5.2 we construct a problem instance of size $\mathcal{O}(r^2)$ that causes both the total sum of ranks and the total length of augmenting paths to be $\Omega(r^3)$. As the construction is quite complicated, we first characterize its basic building blocks, and then demonstrate how to use them to achieve ranks linear in r with an example of size $\mathcal{O}(r^3)$. We end Chapter 5 with a description of how to compress the last instance to achieve the same effect within $\mathcal{O}(r^2)$ size, which proves that the worst-case analysis of Algorithm 3 is tight.

Chapter 1

Preliminaries

In this chapter we introduce the basic notions that form the general framework in which we can formally express the results of this thesis. Unfortunately, the notation and definitions of graph-theoretical concepts vary greatly across the literature. To avoid misunderstandings we intend Preliminaries to be a short reference for the majority of common terms used throughout this thesis. For these which are missing we direct the reader to [27].

For that reason we advise readers who are well-versed in graph theory to skip all the sections of this chapter but for the final part on dynamic matchings. That section is different in that it contains definitions of terms unique to this work, as well as some useful related observations. Moreover, Definition 1.9 specifies the online setting common to all the other chapters, and Definition 1.10 introduces the convenient concept of *minimum surplus*. In other words, we recommend to read Section 1.2.2 in detail, while the other parts of this chapter only if they are unfamiliar.

1.1 Graphs

A graph G is a collection of *vertices* V and *edges* E, formally, it is a pair of sets $G = \langle V, E \rangle$. In this work we assume that both these sets are finite with their cardinalities denoted by $n \stackrel{\text{def}}{=} |V|$ and $m \stackrel{\text{def}}{=} |E|$. If there are two

or more graphs in the context, we refer to the set of vertices and the set of edges of some graph G as $V(G)$ and $E(G)$.

Vertices can be elements of any set, although $\{0, 1, \ldots, n-1\} \subseteq \mathbb{N}$ is one of the most frequently used sets for this purpose. Nevertheless, usually we will just assume the set V as part of the input and refer to the vertices via labels like v_i for the i'th vertex. Edges connect pairs of vertices, however, it is important to decide whether the order of vertices matters or not. If it does, then we get a *directed* graph in which E is a subset of the ordered pairs of vertices $E \subseteq V \times V$—such edges are often called *arcs*. Directed graphs are isomorphic to (finite) binary relations on its set of vertices, in particular, we allow for at most one edge between any pair of vertices in each direction. Moreover, for an edge $e = \langle u, v \rangle$ that leads from u to v, we call v the *head* of e and u the *tail* of e,

$$\begin{aligned} \operatorname{head}(\langle u, v \rangle) &\stackrel{\text{def}}{=} v, \\ \operatorname{tail}(\langle u, v \rangle) &\stackrel{\text{def}}{=} u. \end{aligned}$$

If we were to consider unordered pairs, e.g., $\{u, v\}$ for $u, v \in V$, we get an *undirected* graph. By convention in undirected graphs we disallow loops, that is, edges that connect $v \in V$ to itself, hence

$$E \subseteq \big\{ \{u, v\} \;\big|\; u, v \in V, u \neq v \big\}.$$

Undirected graphs can be though of as symmetric, irreflexive binary relations on V.

We say that e and v are *incident* to each other if v is one of the vertices that e connects. Two edges that are incident to the same vertex, or two vertices that are incident to the same edge are called *adjacent*. Both these relations indicate if two objects are in some sense next to each other, only that incidence relates objects of different kinds (vertices and edges), while adjacency relates two objects of the same kind (two vertices or two edges). Two adjacent vertices can be called also *neighbors* and the set of all vertices adjacent to u is called the *neighborhood* of u. For directed graphs we have

related concepts of *in-* and *out-neighbors*, and *in-* and *out-neighborhood*.

$$\begin{aligned}\mathcal{N}(u) &\stackrel{\text{def}}{=} \{v \in V(G) \mid \{u, v\} \in E(G)\},\\ \mathcal{N}(u \to) &\stackrel{\text{def}}{=} \{v \in V(G) \mid \langle u, v\rangle \in E(G)\},\\ \mathcal{N}(\to u) &\stackrel{\text{def}}{=} \{v \in V(G) \mid \langle v, u\rangle \in E(G)\}.\end{aligned}$$

The above functions extend easily to sets of vertices, for example

$$\mathcal{N}(U) \stackrel{\text{def}}{=} \bigcup_{u \in U} \mathcal{N}(u).$$

A similar notion, the *degree* of a vertex, is the number of edges incident to it. For directed graphs we also distinguish in-degree and out-degree, formally

$$\begin{aligned}\deg(u) &\stackrel{\text{def}}{=} \{e \in E \mid e \text{ is incident to } u\},\\ \deg_{\text{out}}(u) &\stackrel{\text{def}}{=} \{e \in E \mid u \text{ is the tail of } e\},\\ \deg_{\text{in}}(u) &\stackrel{\text{def}}{=} \{e \in E \mid u \text{ is the head of } e\}.\end{aligned}$$

We call a graph G' a *subgraph* of G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. One kind of frequently useful subgraphs are *vertex-induced* subgraphs or simply induced subgraphs—maximal subgraphs defined on some selected subset of vertices of the original graph. More precisely, let $U \subseteq V(G)$ be any set of vertices of G, then the graph induced by U is defined as

$$G[U] \stackrel{\text{def}}{=} \Big\langle U, \{e \in E(G) \mid e \text{ has both its endpoints in } U\}\Big\rangle.$$

Similarly, an *edge-induced* subgraph is defined for a subset of edges $F \subseteq E(G)$ by

$$G[F] \stackrel{\text{def}}{=} \langle V(F), F\rangle.$$

Sometimes a graph (directed or undirected) will be equipped with a function, that labels each edge with its weight, for example $G = \langle V, E, w\rangle$ where $w : E \to \mathbb{N}$. In such case we call G a *weighted* graph.

1.1.1 Paths and Reachability

Although the notion of a path defined here represents the same intuitive object, the exact definition diverges from the one in [27]—here paths consist of *both* edges and vertices.

A *path* is an alternating sequence of edges and vertices such that any two successive objects are incident to each other. Moreover, in the context of directed graphs we require the direction of all the edges to align with the direction of the path, in which case we call the path *directed*. We use the notation $V(P)$ and $E(P)$ to refer, respectively, to the sets of vertices and edges of the path P.

A directed path P from u to v is denoted by $[u \xrightarrow{P} v]$. As our paths contain both edges and vertices, sometimes it is helpful to consider the same path, but excluding its endpoints. We indicate this by a change from a square bracket "[" to a round bracket "(". For example, $[u \xrightarrow{P'} v)$ means a path P' such that $u \in V(P')$, but $v \notin V(P')$.

Path is *simple* if it does not visit any vertex twice, including both its endpoints. A path that starts and ends at the same vertex is called a *cycle*, where a simple cycle means a cycle that does not visit twice any vertex other than its endpoint. The *length* of a path or a cycle is the number of edges in it.

If a vertex v belongs to a path P, then we can split the path using that vertex, for example,

$$[u_1 \xrightarrow{P_1} v] \circ (v \xrightarrow{P_2} u_2) = [u_1 \xrightarrow{P} u_2).$$

Here, by "$\circ$" we denote the concatenation operator, which reverses the splitting—it joins two paths with a common endpoint, and common direction if the involved paths are directed. Any initial part of the path, i.e., continuous fragment containing its beginning, is called its *prefix*. Similarly, a *suffix* of P is a terminal part of path P.

For an undirected graph G we say that u and v are *connected* if there exists a path from u to v. It is easy to check that *connectedness* is an equivalence relation. Its equivalence classes form the *connected components* of G.

The analogous relation in the context of directed graphs is called *reachability*, where v is *reachable* from u if there exists a directed path from u to v.

1.1.2 Special Classes of Graphs

Many graph problems that are hard in the general case, become much easier if they are considered for graphs that satisfy some additional properties. Two classes that are of special interest to us are *trees* and *bipartite graphs*.

Trees We call an undirected graph a *tree* if it is acyclic and connected. In particular, in any tree there is exactly one simple path between any pair of vertices. Frequently one of the vertices is distinguished and becomes the *root* of the tree. In such case we say that the tree is *rooted* in v. Trees have their own terminology, the list below contains the most important concepts (see Figure 1.1). Let v and u be any two vertices of a tree, then

- v is a *leaf* if it is of degree one;
- u is an *ancestor* of v if u belongs to the path that connects v and the root, in particular the root is an ancestor of any other vertex, yet it is not a *proper ancestor* of itself;
- u is a *parent* of v if it is the direct ancestor of v, i.e., the first vertex on the path to the root;
- u is a *descendant* of v if v is an ancestor of u;
- u is a *child* of v if v is a parent of u;
- for any graph G, a *subtree* of G is any subgraph that is itself a tree;
- if G is a rooted tree, then the *subtree rooted in* v is a subtree that consists of v and all its descendants;
- if G is an unrooted tree, then a *subtree of* v is just any connected component of $G \setminus \{v\}$.

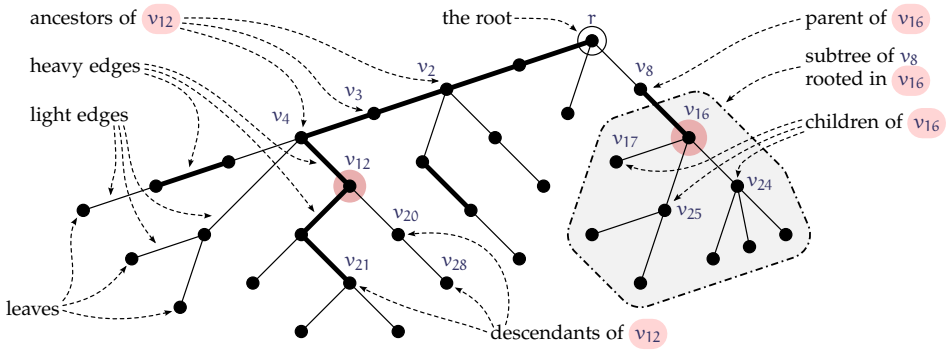

Figure 1.1: A rooted tree illustrating the basic terminology. Heavy edges and light edges constitute the heavy-light decomposition of a tree.

One characteristic of trees that we will use in Chapter 2 is that they are amenable to the heavy-light decomposition, a technique introduced by Sleator and Tarjan in [90].

Consider a tree G rooted at $r \in V(G)$, and partition its edges into heavy and light, depending on whether the size of the subtree is strictly bigger than half of the size of the subtree rooted at parent. Formally, let $v \in V(G)$ be any vertex other than root r and set $p_v \in V(G)$ to be its parent, then

$$\text{edge } \{v, p_v\} \in E(G) \text{ is } \textit{heavy} \quad \overset{\text{def}}{\iff} \quad \left|\mathrm{subtree}(v)\right| > \tfrac{1}{2}\left|\mathrm{subtree}(p_v)\right|,$$

while non-heavy edges we call *light*. In the diagram above the heavy and light edges were marked respectively with thick and thin lines. Observe, that because of the size requirements, each time we traverse a light edge away from the root r, the size of the current subtree halves. In other words, for any vertex $v \in V(G)$ there are at most $\lfloor \log_2 |G| \rfloor$ light edges on the simple path from r to v. Moreover, each vertex can have at most two heavy incident edges, thus heavy edges form vertex-disjoint paths. However, paths are of much simpler structure than arbitrary trees, hence allow for more efficient handling despite being possibly numerous.

Thanks to that characteristic of trees Sleator and Tarjan were able to achieve an amortized logarithmic complexity for their dynamic link/cut tree structure. This is also the reason why we use the heavy-light decomposition in this thesis. Although the actual method is slightly different—we

consider an edge heavy if it leads to the biggest subtree, i.e., each non-leaf vertex has at least one heavy incident edge (see Lemma 2.12 for details)—the general underlying idea is the same.

Bipartite graphs A graph is called *bipartite* if its set of vertices can be partitioned into two disjoint subsets such that no edge is contained in only one of them. Equivalently, a graph is bipartite if and only if it has no cycles of odd length. For simplicity, we call these two sets of vertices *black* and *white*, denoted respectively by B and W. Although a bipartite graph may admit more than one partition, that is, a graph of c connected components has 2^c possible bipartitions, for our analysis, it does not matter—we assume the partition is a part of the input. An example of a bipartite graph is presented in Figure 1.2.

1.2 Matchings

Given a graph G, a *matching* is a subset of edges of G that are pairwise not adjacent, i.e., they share no vertices. A matching is *maximal* if it is maximal with respect to inclusion, while *maximum* matching is one of maximum cardinality. We say that vertices incident to M are *matched* or M-*saturated*, and we call M *perfect* if it matches all the vertices of V, that is, it is of size $\frac{1}{2}|V|$. Vertices that are not incident to M are *unmatched* or M-*free*. If the matching M is clear from context, the prefix "M-" will be skipped.

Given a matching M in a graph G we say that a path A in G is M-*alternating* if exactly its every second edge belongs to M, that is, for any two consecutive edges e_i, e_{i+1} on A precisely one of them is in M. Furthermore, a path is M-*augmenting* if it is a simple M-alternating path that both starts and ends at M-free vertices. Such a path has to be of odd length.

Observation 1.1. For two arbitrary matchings M_1 and M_2 in G, the edge-induced graph $G[M_1 \cup M_2]$ consists of

- cycles C such that $|E(C) \cap M_1| = |E(C) \cap M_2|$,

- simple paths P such that

$$\begin{aligned} \left|E(P) \cap M_1\right| &\leqslant \left|E(P) \cap M_2\right| + 1, \\ 1 + \left|E(P) \cap M_1\right| &\geqslant \left|E(P) \cap M_2\right|. \end{aligned}$$

Proof. For any vertex v of $G[M_1 \cup M_2]$ we have $1 \leqslant \deg(v) \leqslant 2$. Moreover, any two adjacent edges belong to different matchings, so for each cycle we have to have an equal number of edges from M_1 and from M_2, while for paths the difference can be at most one. □

The above observation holds also when we use the symmetric difference $\oplus$ instead of just set union $\cup$ of matchings.

Observation 1.2. The claim of Observation 1.1 holds also for the edge-induced graph $G[M_1 \oplus M_2]$. ▫

Theorem 1.3 (Berge's lemma [12])**.** *Matching M is maximal if and only if there is no M-augmenting path in G.*

Proof. Let A be an M-augmenting path in G, then $M' = M \oplus A$ is a matching of bigger cardinality. Conversely, let M' be a matching of strictly bigger cardinality than M, then $G[M' \cup M]$ has to contain a component that has strictly more edges of M' than edges of M, in which case, by Observation 1.1 it has to be a simple M-alternating path of odd length, thus an M-augmenting path. □

Observation 1.4. Let M_{OPT} be a matching of maximum cardinality in graph G and M be any matching also in G. If the length of the shortest path augmenting M is at least k, then

$$|M| \geqslant |M_{OPT}|\left(1 - \frac{2}{k}\right).$$

Proof. Consider the matching-induced graph $G\left[M_{OPT} \oplus M\right]$, it contains $\left(|M_{OPT}| - |M|\right)$ disjoint M-augmenting paths, each of length at least k.

Moreover, every vertex of such path is a vertex of M_{OPT}. Hence,

$$\Big(|M_{OPT}| - |M|\Big)k \leqslant 2|M_{OPT}|$$

and the observation follows. □

1.2.1 Matchings in Bipartite Graphs

Matchings in bipartite graphs, thanks to the lack of odd-length cycles, have quite elegant structure that manifests itself strongly in the following three results.

Let $G = \langle B \uplus W, E\rangle$ be an undirected bipartite graph and M a matching in G. Observe that each edge has one black and one white vertex. That allows us to encode if the edge is matched in M using its direction. Define $G^{\bowtie M} \stackrel{\text{def}}{=} \langle B \uplus W, E^{\bowtie M}\rangle$ as the directed graph on the vertices of G in which we orient matched edges from white vertices to black, and unmatched edges from black to white, formally

$$\begin{aligned} E^{\bowtie M} \stackrel{\text{def}}{=} &\left\{\langle w, b\rangle \;\middle|\; w \in W,\ b \in B,\ \{w, b\} \in M\right\} \\ &\cup \left\{\langle b, w\rangle \;\middle|\; w \in W,\ b \in B,\ \{w, b\} \in E \setminus M\right\}. \end{aligned}$$

The above idea is illustrated in the first diagram of Figure 1.2: we direct all the matched edges to the right, and all the unmatched edges to the left. In the second picture, which presents the same graph, we can notice an immensely useful feature of $G^{\bowtie M}$, that is, the M-alternating paths in G are exactly the directed paths of $G^{\bowtie M}$.

Observation 1.5. A path A is an alternating path in G if and only if $A^{\bowtie M}$ is a directed path in $G^{\bowtie M}$.

Proof. Let e_1 and e_2 be any two consecutive edges of A, ${e_1}^{\bowtie M}$ and ${e_2}^{\bowtie M}$ be their counterparts in $A^{\bowtie M}$, and v be the common vertex between them. Observe that counterparts of matched edges in $G^{\bowtie M}$ lead always to B, while counterparts of unmatched edges lead always to W.

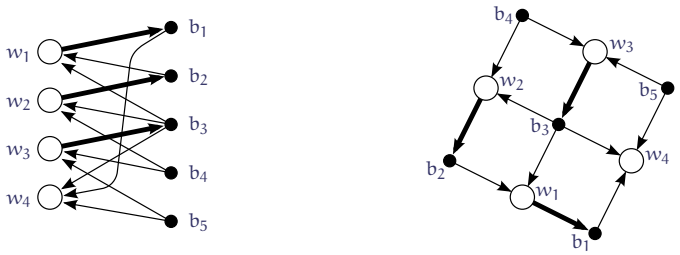

Figure 1.2: A matching-directed bipartite graph: matched edges are directed left-to-right, while unmatched edges are directed right-to-left. The M-alternating paths in G are exacly the directed paths of $G^{\bowtie M}$.

Necessity ($\Rightarrow$). The above implies that v, which is either black or white, is the head of exactly one edge of ${e_1}^{\bowtie M}$ and ${e_2}^{\bowtie M}$, and is the tail of exactly one of these edges. In other words, any vertex of $A^{\bowtie M}$ other than its endpoints is the tail and the head of exactly one edge of $A^{\bowtie M}$. This implies that $A^{\bowtie M}$ is a directed path.

Sufficiency ($\Leftarrow$). Conversely, if v is the head and the tail of exactly one edge of ${e_1}^{\bowtie M}$ and ${e_2}^{\bowtie M}$ then exactly one of them leads to B while the other leads to W. This in turn implies that exactly one of e_1 and e_2 belongs to M and proves A to be an alternating path. □

Observation 1.6. If A is an M-alternating path, then there exists a simple M-alternating path A' that has the same endpoints and $A' \subseteq A$.

Proof. We prove that we can remove any cycle from A and it will still remain an alternating path. As A is of finite length, this way we can remove all the cycles, and thus construct a simple alternating path.

Let A be an alternating path that visits $v \in V$ at least twice, by Observation 1.5 $\left[u_1 \xrightarrow{A^{\bowtie M}} u_2\right)$ is a directed path in $G^{\bowtie M}$ which also visits v at least twice. Set $\left[u_1 \xrightarrow{{A_1}^{\bowtie M}} v\right)$ to be the longest prefix of $A^{\bowtie M}$ that does not contain v and $\left[v \xrightarrow{{A_2}^{\bowtie M}} u_2\right)$ to be the shortest suffix that does contain v. Then, the concatenation of ${A_1}^{\bowtie M}$ and ${A_2}^{\bowtie M}$ is a directed path that connects u_1 to u_2, so $A_1 \circ A_2 \subseteq A$ is an alternating path in G with the same endpoints as A. □

Theorem 1.7 (Hall's theorem [57]). *A bipartite graph G contains a matching M that matches all vertices of $X \subseteq V$ if and only if it satisfies Hall's condition*

$$\forall Y \subseteq X.\ |Y| \leqslant |\mathcal{N}(Y)|. \qquad \text{(Hall's condition)}$$

Originally this statement was worded in terms of sets and their representants, in particular, there would be an implicit assumption that $X \subseteq B$ (or $X \subseteq W$, by symmetry there is no difference). That assumption is also prevalent in the statement of Hall's theorem in graph-theoretic language, and it is covered here by *Case 1* and *Case 2* of the following proof. However, we give here a slightly stronger version of Hall's theorem—one that does not need the aforementioned assumption—by reducing the problem in *Case 3*.

Proof. ***Necessity of Hall's condition*** $(\Rightarrow)$**.** Let M be the matching that matches all vertices of X and set $f : X \to V$ to be a function that assigns any vertex $x \in X$ to its matched pair in M, formally

$$f(x) = v \iff \{x, v\} \in M.$$

Obviously $f(x) \in \mathcal{N}(x)$. Moreover, the edges in M do not share vertices, so f is injective and

$$\forall Y \subseteq X.\ |Y| = |f(Y)| \leqslant |\mathcal{N}(Y)|.$$

Sufficiency of Hall's condition $(\Leftarrow)$**.** We proceed by induction on $|X|$. Clearly, the theorem is true for $|X| = 1$, in the following we assume that $|X| \geqslant 2$ and that the theorem is true for all sets of strictly smaller cardinality.

Case 1. Suppose that $X \subseteq B$ or $X \subseteq W$ and that we have a positive slack for all proper non-empty subsets of X, that is

$$\forall Y \subsetneq X, Y \neq \varnothing.\ |Y| + 1 \leqslant |\mathcal{N}(Y)|.$$

Let e be an arbitrary edge incident to X and set $G' = G[V \setminus V(e)]$. Observe that in G' we remove exactly one vertex from both B and W, so $\mathcal{N}_{G'}(Y)$ can decrease at most by one with respect to $\mathcal{N}_G$. In other words, G' satisfies Hall's condition, so we can find a matching M' that covers $V(G') \cap X$ and then extend it to $M = M' \cup \{e\}$.

Case 2. Suppose that $X \subseteq B$ or $X \subseteq W$ and that there exists a proper non-empty subset Y of X such that

$$\big|Y\big| = \big|\mathcal{N}(Y)\big|.$$

Clearly $G' = G\big[Y \cup \mathcal{N}(Y)\big]$ satisfies Hall's condition. However, for any $Z \subseteq X \setminus Y$ we have

$$\begin{aligned}\big|Y\big| + \big|Z\big| = \big|Y \cup Z\big| &\leqslant \big|\mathcal{N}(Y \cup Z)\big| \\ &= \big|\mathcal{N}(Y) \cup \big(\mathcal{N}(Z) \setminus \mathcal{N}(Y)\big)\big| \\ &= \big|\mathcal{N}(Y)\big| + \big|\mathcal{N}(Z) \setminus \mathcal{N}(Y)\big|,\end{aligned}$$

so $G'' = G\big[\big(X \setminus Y\big) \cup \big(\mathcal{N}(X) \setminus \mathcal{N}(Y)\big)\big]$ satisfies Hall's condition as well. Observe that $V(G')$ and $V(G'')$ are disjoint, hence, we can set $M = M' \cup M''$ where M' and M'' are respectively Y-saturating and $(X \setminus Y)$-saturating matchings obtained from G' and G''.

Case 3. Finally, suppose that X contains vertices from both B and W, more formally

$$X \cap B \neq \varnothing \quad \text{and} \quad X \cap W \neq \varnothing.$$

Because $|X \cap B| < |X|$ and $|X \cap W| < |X|$, we know that there are matchings M_B and M_W that match all the vertices of $X \cap B$ and $X \cap W$ respectively. The idea is to pick, for each connected component of $M_B \cup M_W$, the edges that saturate as many vertices of X as possible.

Consider a connected component C of $M_B \cup M_W$ and let

$$C_\alpha = V\big(E(C) \cap \alpha\big) \cap X \quad \text{for any} \quad \alpha \subseteq E.$$

Observe that no path of $M_B \cup M_W$ can have both endpoints outside of X,

so either $C_{M_B} \subseteq C_{M_W}$ or $C_{M_B} \supseteq C_{M_W}$. Therefore we can set

$$M = \bigcup_{\text{connected component } C \text{ of } M_B \cup M_W} \text{extract}(C),$$

where

$$\text{extract}(C) = \begin{cases} E(C) \cap M_B & \text{if } \left|C_{M_B}\right| \geqslant \left|C_{M_W}\right|, \\ E(C) \cap M_W & \text{otherwise.} \end{cases}$$

□

The Hall's condition is a convenient tool to work with. However, it is too imprecise—it only tells us whether there is a matching or not, conflating all the different graphs into only two classes. In a prelude to Section 1.2.2, consider what happens when we add a vertex to a graph, in particular, how the inequalities in Hall's condition continue or cease to hold. There is a notion that could help us in this matter, a well-known generalization of Hall's condition, namely Hall's condition *with surplus*. Intuitively it describes how many vertices we could add and still be guaranteed to have a saturating matching.

Definition 1.8. We say that set X satisfies Hall's condition *with surplus* k if $k \in \mathbb{Z}$ is the biggest (possibly negative) integer such that

$$\forall Y \subseteq X, Y \neq \varnothing.\ \left|Y\right| + k \leqslant \left|\mathcal{N}(Y)\right|.$$

◇

Nevertheless, the definition is concerned with the worst case—what would happen if we were to add vertices in the least favorable configuration. On the other hand, we would like to analyze what happens when we add neighbors to some particular vertex, and thus need an even more fine-grained approach. To this end, in the next section we introduce the concept of the *minimum surplus* of a set of vertices.

1.2.2 Dynamic Matchings

The main result of this thesis assumes a dynamic setting, that is, one in which the algorithm maintains a solution as the underlying structure changes. The challenge is to devise a way to reuse some already computed information, e.g., the old solution, to obtain the new solution in a smarter way than recalculating it from scratch every time.

There are many possible ways in which a graph could change over time, for example, given a graph G we might want to add a new vertex $v \notin V(G)$, and two new edges $e_1, e_2 \notin E(G)$. More precisely, we would like to construct graph $G' = \langle V', E' \rangle$ where $V' = V(G) \cup \{v\}$ and $E' = E(G) \cup \{e_1, e_2\}$. To keep track of all the changes easily, instead of modifying a single graph G we consider a sequence of graphs $G_0, G_1, \ldots$ where $G_0 = G$ and subsequent elements represent the successive updates done to G.

The algorithms described in Chapters 2 and 4 maintain a maximum cardinality matching, which, obviously, may change between turns. Therefore, instead of modifying the current matching M, we use a sequence of matchings $M_0, M_1, \ldots$, analogously to the sequence of graphs described above.

Intuitively, in this thesis we assume an incremental setting in which we are given the set W up front, and then each turn a vertex of B arrives, together with all its incident edges. At all times we maintain a maximum matching in the graph. Formal definition is as follows.

Definition 1.9. The input to the algorithm is a sequence of bipartite graphs $G_0, G_1, \ldots, G_n$ such that

- $G_0 = \langle W \uplus B_0, E_0 \rangle = \langle W \uplus \varnothing, \varnothing \rangle$,
- $G_t = \langle W \uplus B_t, E_t \rangle = \langle W \uplus B_{t-1} \cup \{b_t\},\ E_{t-1} \cup E_{b_t} \rangle$ where

$$E_{b_t} \neq \varnothing, \qquad E_{b_t} \subseteq \Big\{ \{b_t, w\} \;\Big|\; w \in W \Big\}.$$

The output of the algorithm is a sequence of matchings $M_0, M_1, \ldots, M_n$ such that M_t is a matching in G_t which depends only on graphs encoun-

tered up to t, that is, mapping

$$\left(G_0, G_1, \ldots, G_t\right) \to M_t$$

is a function for any $1 \leqslant t \leqslant n$ and any input sequence $G_0, G_1, \ldots, G_n$. $\diamond$

Note, that $E_{b_t} \neq \varnothing$. The reason is that, although white isolated vertices might have some neighbors in future turns, black isolated vertices are obviously impossible to be matched. Hence, to avoid special cases, we can exclude them from analysis without any loss of generality.

Throughout this text we use phrase "at time t" or "at turn t", which formally means "in $G_t^{\bowtie M_{t-1}}$", that is, in a graph with the new vertex b_t added, but before the new matching M_t has been calculated and applied.

Definition 1.10. Consider a sequence of bipartite graphs $G_0, G_1, \ldots$ and respective matchings $M_0, M_1, \ldots$

- For any matched white vertex $w \in W(M_t)$ we denote its matched pair in M_t by b_w^t.

- For any turn t we define the set of *seeds* $\mathcal{S}_t$ as the set of M_t-free white vertices of G_t,
$$\mathcal{S}_t \stackrel{\text{def}}{=} W(G_t) \setminus W(M_t).$$

- We define the *minimum surplus* of $X \subseteq V(G)$ as
$$\min \operatorname{surplus}_G(X) \stackrel{\text{def}}{=} \min_{Y \in \mathcal{Y}} \left|\mathcal{N}_G(Y) \cup \left(X \cap W(G)\right)\right| - \left|Y\right|$$
where $\mathcal{Y}$ is a collection of sets $Y \subseteq B(G)$ that are supersets of $X \cap B(G)$ and satisfy Hall's condition, i.e.,
$$\mathcal{Y} = \left\{ Y \subseteq B(G) \;\middle|\; Y \supseteq X \cap B(G),\ \forall Z \subseteq Y.\ \left|Z\right| \leqslant \left|\mathcal{N}_G(Z)\right| \right\}.$$
Intuitively, in a sense, it is the minimum surplus (see Definition 1.8) of any reasonable superset of X, in particular, the above is a proper definition only if $X \cap B(G)$ satisfies Hall's condition.

- We consider vertex v *alive* if $\min \operatorname{surplus}(\{v\}) > 0$, otherwise v is *dead*. We denote the set of dead vertices of G by $\mathcal{D}(G)$ and $\mathcal{D}_t \stackrel{\text{def}}{=} \mathcal{D}(G_t)$. $\diamond$

Lemma 1.11. *For any bipartite graph G and vertex $v \in V(G)$ the four following conditions are equivalent:*

(1) *v is alive;*

(2a) *if $v \in W$, then there exists a maximum matching M such that v is M-free;*

(2b) *if $v \in B$, then there exists a maximum matching M such that v has an M-free neighbor;*

(3) *there exists a maximum matching M and an M-free white vertex $s \in W$ such that there exists a simple alternating path between v and s, i.e., a simple directed path from v to a seed s in $G^{\bowtie M}$;*

(4) *for any matching M there exists M-free white vertex $s \in W$ such that there exists a simple alternating path between v and s, that is, a simple directed path from v to s in $G^{\bowtie M}$.*

Proof. ***Case*** $(1) \implies (2a)$**.** If $v \in W$, then let $G' = G \setminus \{v\}$ and observe that for any $X \subseteq B$ if X satisfied Hall's condition in G, then it satisfies Hall's condition in G'. That is because if $v \in \mathcal{N}_G(X)$, then Hall's condition was satisfied with surplus of at least 1. Hence, for any matchinig M in G we can find a matching M' in G' of the same cardinality. As G' is a subgraph of G, M' happens to be also a matching in G. However, $v \notin V(G')$, so M' is a maximum cardinality matching in G that does not match v.

Case $(1) \implies (2b)$**.** Take any set $X \subseteq B$ that satisfies Hall's condition. Then for any $Y \subseteq X$ we have that $\mathcal{N}(v) \setminus \mathcal{N}(Y) \neq \varnothing$ implies

$$\left|\mathcal{N}(Y) \cup \mathcal{N}(v)\right| - \left|Y \cup \{v\}\right| \geqslant 0.$$

and $\mathcal{N}(v) \subseteq \mathcal{N}(Y)$ implies

$$\begin{aligned}\big|\mathcal{N}(Y) \cup \mathcal{N}(v)\big| - \big|Y \cup \{v\}\big| &\geqslant \big|\mathcal{N}(Y)\big| - \big|Y\big| - 1 \\ &\geqslant \min \operatorname{surplus}\big(\mathcal{N}(v)\big) - 1 \\ &\geqslant \min \operatorname{surplus}\big(\{v\}\big) - 1 \\ &\geqslant 0.\end{aligned}$$

However, this makes $X \cup \{v\}$ a valid candidate for the minimum surplus of $\{v\}$, so $\big|\mathcal{N}(Y) \cup \mathcal{N}(v)\big| - \big|Y \cup \{v\}\big| \geqslant 1$ for all $Y \subseteq X$. Therefore, within graph G' defined as

$$G' = \big\langle V \cup \{b\}, E \cup \big\{\{b, w\} \bigm| w \in \mathcal{N}(v)\big\}\big\rangle$$

we can infer from the last inequality that if X satisfies Hall's condition, then both $X \cup \{v\}$ and $X \cup \{v, b\}$ also satisfy Hall's condition. In other words, there exists a maximum matching M in G' that matches both v and b, but then M constrained to G is a matching that does not match a neighbor of v.

Case $(2) \implies (3)$**.** Trivially, since any path P of length $|P| \leqslant 1$ is an M-alternating path. For $v \in W$ we set $s = v$ and for $v \in B$ we set $s = w$, where w is the M-free neighbor of v.

Case $(3) \implies (4)$**.** Let M be a maximum matching and $s \in W$ be an M-free vertex such that there exists a simple alternating path $\big[v \overset{P}{\longleftrightarrow} s\big]$. If $v \in W$, then set $w = v$, or otherwise set w to be the matched pair of v if $v \in B$. In the latter case v has to be matched, because otherwise P would be an augmenting path and M could not be a maximum matching.

Consider $G^b = \langle V(G) \cup \{b\}, E(G) \cup \{b, w\}\rangle$ for a new black vertex b, and observe that G^b admits an M-augmenting path $A = \big[b \overset{\{b,w\}\cup P}{\longleftrightarrow} s\big]$, construct $M^b = M \oplus A$.

Now take any matching M' in G. Observe that M' is a non-maximum matching in G^b, so M^b, a matching that saturates b, is of strictly greater cardinality than M'. Thus, there exists M'-augmenting path A' in G^b that

matches b, which is one the connected components of $M' \oplus M^b$. However, that means $A' \setminus \{b, w\}$ is an alternating path to some white M'-free vertex s' that starts with a matched edge, and thus can be easily extended to an alternating path from v to s' in G.

Case $(4) \implies (1)$**.** Set $X = \{v\}$. Take any set $Y \subseteq B$ that satisfies Hall's condition and $v \in Y$. Note that $v \in Y$ is equivalent to $Y \supseteq (X \cap B) = \{v\}$, in other words, Y is any candidate set for the minimum surplus of $X = \{v\}$. By Hall's theorem we know there exists a matching M that saturates all the vertices of Y, and that $M' = \{e \in M \mid e \text{ is incident to } Y\}$ is also a matching, which happens to be of size $|Y|$.

Moreover, by (4) there is an M'-alternating path that connects v to some M'-free white vertex s. Furthermore, because $Y \supseteq (X \cap B)$, we have that either v is incident to M', but then $s \in \mathcal{N}(Y)$ and

$$\left|\mathcal{N}(Y) \cup (X \cap W)\right| \geqslant \left|\mathcal{N}(Y)\right| \geqslant \left|Y\right| + \left|\{s\}\right| = \left|Y\right| + 1,$$

or v is white and non-incident to M', i.e., $s = v$ and

$$\left|\mathcal{N}(Y) \cup (X \cap W)\right| = \left|\mathcal{N}(Y) \uplus \{v\}\right| \geqslant \left|Y\right| + 1. \qquad \square$$

Observation 1.12. If vertex v became dead at some turn, then it will stay that way until the end of the algorithm. ▫

Up to this point the matchings M_t could change arbitrarily between the turns. Nevertheless, all the algorithms we consider in the following chapters use the augmenting paths approach. Henceforth, we assume that the algorithm maintains the matchings using augmenting paths, which we denote by A_t. Formally

$$A_t = M_t \oplus M_{t-1} \quad \text{for} \quad 1 \leqslant t \leqslant n.$$

It is true that augmenting paths constrain the algorithm in what it can do. However, such an assumption has a number of consequences

that make the analysis simpler. For example, an augmenting path cannot *un*match a vertex—it can only reassign it to some other pair. This will become helpful in Chapter 3, because it implies the sets of seeds $\mathcal{S}_t$, i.e., the set of M_t-free white vertices, satisfy $\mathcal{S}_{t+1} \subseteq \mathcal{S}_t$.

Observation 1.13. A vertex matched at turn t stays matched until the end of the algorithm,

$$V(M_t) \subseteq V(M_{t+1}) \quad \text{for} \quad 1 \leqslant t < n. \qquad \square$$

Corollary 1.14. Once a vertex ceased to be a seed, it cannot become one again,

$$\mathcal{S}_{t+1} \subseteq \mathcal{S}_t \quad \text{for} \quad 1 \leqslant t < n. \qquad \square$$

Another characteristic of augmenting paths is that *dead* vertices are insignificant—Lemma 1.11 implies that any vertex other than b_t that belongs to augmenting path A_t at turn t had to be alive at turn $(t-1)$. Conversely, when some vertex becomes *dead*, then from that turn no augmenting path will ever use it, in which case we can just remove it from the graph. The only exception is the vertex b_t that has just arrived. Even if it is dead at the end of turn t, it is still possible for the path A_t to match it. This is because A_t is an M_{t-1}-augmenting path, and M_{t-1} may not be a maximum matching in G_t.

Observation 1.15. If a M_t-free vertex is dead at turn t, then it will stay unmatched until the end of the algorithm. In particular, there is a maximum cardinality matching that does not match that vertex. $\square$

Corollary 1.16. If b_t cannot be match at the turn t, when it arrives, without loss of generality we can remove it from the graph. $\square$

Chapter 2

On the Shortest Augmenting Paths Approach

In this chapter we consider a classic approach to the maximum matching and maximum flow problem, namely the shortest augmenting paths, and we try to apply it to the dynamic setting described in Definition 1.9. In other words, we attempt to match each new vertex b_t that arrives, using the shortest augmenting path available at turn t. Formally, we require the path $A_t = M_{t-1} \oplus M_t$ to be a shortest M_{t-1}-augmenting path in G_t.

Although the shortest paths approach is one of the basic algorithmic techniques, it is far from being fully understood, even in the context of matchings and flows. In the following sections we present three ideas that shed some additional light on the problem.

One of the challenges related to the dynamic setting is the evolving nature of the input graph. To make the potential analysis easier, in Section 2.1 we introduce a reduction that transforms any bipartite graph into an instance, in which the original structure is static and available from the beginning. Not only it can be used to bound the total length of augmenting paths, but we believe it preserves all the essential characteristics of the graph, including the min surplus of any subset of vertices of the original final graph.

In the second section we investigate the total length of the augmenting paths found by an algorithm described in the first paragraph. Chaudhuri,

Daskalakis, Robert and Lin [24] conjecture that the total length of augmenting paths found by such an algorithm is $\mathcal{O}(|V| \log |V|)$. Still, to the best of author's knowledge, no better bound than the trivial $\mathcal{O}(|V|^2)$ is known even for the special case of trees. For that particular case we improve the bound to $\mathcal{O}(|V| \log^2 |V|)$.

Finally, in the last section we focus on a rather surprising characteristic of a proof of the bound from Section 2.2—it does not depend on the matching it uses, only on the structure of the tree and which vertices are alive or dead. Thus, we end this chapter with a conjecture about the behavior of the shortest augmenting paths algorithm in a setting where an adversary modifies the calculated matching after each applied augmenting path.

2.1 General Bipartite Graphs

As mentioned in the introduction above, one of the problems in an online setting is the changing nature of the graph. For example, the assumption that one part of the graph is bigger than some other part may cease to be true if enough vertices are added to the latter. Therefore, we introduce a reduction which may greatly simplify the algorithm analysis if it suffers from such issues.

The basic idea is simple—instead of adding new vertices and edges to the graph, we incorporate the whole structure from the very beginning, and then only "switch on" the parts that should become available. We do this by supplying each original black vertex b_t with a new white vertex w'_t, both present in the graph from the start. We assume that all the b_t's begin matched to their new pairs, so that their presence starts to matter only at turn t, after another black vertex b'_t claims w'_t for itself.

That means that the original structure of the graph, including all the vertices b_t and w'_t for $1 \leqslant t \leqslant n$, is present from the very first turn. The only changes that are made are the additions of black vertices b'_t, which are of degree 1. Formally, we define the transformed graphs G'_t as follows.

Definition 2.1. For any input sequence of graphs $G_0, \ldots, G_n$ and output sequence of matchings $M_0, \ldots, M_n$ as defined in Definition 1.9, construct

sequences $G'_0, \ldots, G'_n$ and $M'_0, \ldots, M'_n$, as follows

$$
\begin{aligned}
G'_t &= \langle W' \uplus B'_t, E'_t \rangle, & W' &= W(G_n) \cup \{w'_1, w'_2, \ldots, w'_n\}, \\
B'_0 &= B(G_n), & E'_0 &= E(G_n) \cup \big\{\{b_i, w'_i\} \;\big|\; 1 \leqslant i \leqslant n\big\}, \\
B'_t &= B'_{t-1} \cup \{b'_t\}, & E'_t &= E'_{t-1} \cup \big\{\{w'_t, b'_t\}\big\}, \\
M'_t &= M_t \cup \big\{\{b'_i, w'_i\} \;\big|\; 1 \leqslant i \leqslant t, b_i \in V(M_t)\big\} \\
&\quad \cup \big\{\{b_i, w'_i\} \;\big|\; t < i \leqslant n \;\text{ or }\; b_i \notin V(M_t)\big\}. & & \diamond
\end{aligned}
$$

Figure 2.1 presents an example graph G and the transformed instance G' at turn $t = 3$. The red hue indicates the shortest augmenting paths for vertices b_3 and b'_3 respectively, both of which have been marked with an additional circle. It is worth noting that vertex w'_i is related to b_i, not w_i, in particular, there exists vertex w'_5 despite the fact that the original graph G has only four white vertices.

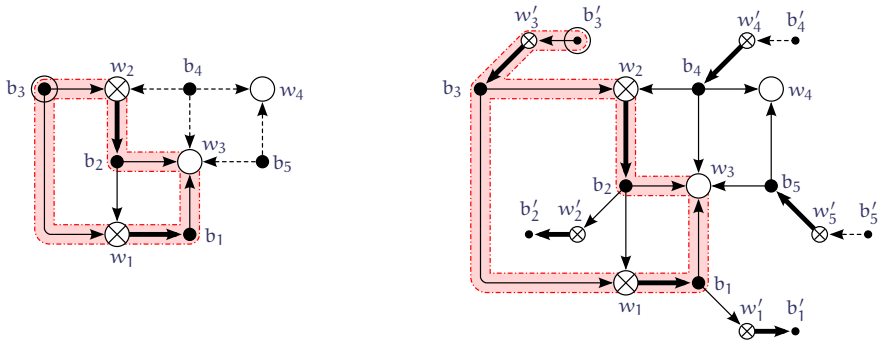

Figure 2.1: The original problem instance (left) and its reduction (right) at turn $t = 3$. Each black vertex b_i has two auxiliary vertices: w'_i and b'_i. The shortest augmenting paths from vertices respectively b_3 and b'_3 are indicated by a light red background. The dashed lines are the edges of black vertices that have not been added yet.

Observation 2.2. For the sequences of graphs and matchings defined above, a path $\left[b_t \overset{A}{\leftrightarrow} s_t\right]$ is an M_{t-1}-augmenting path in G_t if and only if $A' = \left[b'_t \overset{b'_t w'_t b_t}{\longleftrightarrow} b_t\right) \circ \left[b_t \overset{A}{\leftrightarrow} s_t\right]$ is an M'_{t-1}-augmenting path in G'_t. $\square$

Corollary 2.3. For the above defined sequences, $M_0, \ldots, M_n$ is an output of some run of the shortest augmenting path algorithm on $G_0, \ldots, G_n$ if and only if $M'_0, \ldots, M'_n$ is an output of some run of the shortest augmenting path algorithm on $G'_0, \ldots, G'_n$.

Proof. We proceed by induction on t. Clearly, the claim is true when $t = 0$, that is, if neither algorithm performed any augmentations. To prove the inductive step we assume that the theorem is true at the previous turn.

Let $\mathcal{A}$ be the collection of all M_{t-1}-augmenting paths in G_t and $\mathcal{A}'$ be the collection of all M'_{t-1}-augmenting paths in G'_t. As both M_{t-1} and M'_{t-1} are maximum matchings in their respective graphs, then all paths in $\mathcal{A}$ start with b_t and all paths in $\mathcal{A}'$ start with b'_t. Therefore, by Observation 2.2 there is a one-to-one correspondence $\phi(A) = \left[b'_t \xleftrightarrow{b'_t w'_t b_t} b_t\right) \circ A$ between $\mathcal{A}$ and $\mathcal{A}'$.

If both collections of augmenting paths are empty, then both $b_t \in V(G_t)$ and $b'_t \in V(G'_t)$ will remain unmatched, and the claim is true at t. Otherwise, observe that $|\phi(A)| = |A| + 2$, so for any $A \in \mathcal{A}$ we have that A is a shortest M_{t-1}-augmenting path if and only if $A' = \phi(A)$ is a shortest M'_{t-1}-augmenting path in G'_t. It suffices to verify that $M'_t = M'_{t-1} \oplus \phi\big(M_t \oplus M_{t-1}\big)$, which is indeed the case due to $M_{t-1} = M'_{t-1} \cap E(G_t)$ and $\{w'_t, b_t\} \in M'_{t-1}$. □

Corollary 2.4. Let ϕ be the total length of augmenting paths of the shortest augmenting path algorithm on $G_0, \ldots, G_n$ starting from M_0. Then, the total length of all augmenting paths of the shortest augmenting path algorithm on $G'_0, \ldots, G'_n$ starting from M'_0 is equal to $\phi + 2n$. ▫

In fact, the connection between $G_0, \ldots, G_n$ and $G'_0, \ldots, G'_n$ is a bit deeper than Corollary 2.4 suggests. The intuition behind the next lemma is that, with respect to incremental matchings, the structure of the original sequence of graphs and their counterparts are essentially the same.

Lemma 2.5. *For all $X \subseteq V(G)$ such that $X \cap B(G)$ satisfies Hall's condition in G we have*

$$\min \operatorname{surplus}_{G'}(X) = \min \operatorname{surplus}_{G}(X).$$

Proof. ***Case*** $(\leqslant)$**.** Take any $Z \subseteq B(G)$, and set $Z' = Z \cup \{b'_i \mid b_i \in Z\}$, then

$$\begin{aligned} \left|\mathcal{N}_{G'}(Z')\right| - \left|Z'\right| &= \left|\mathcal{N}_G(Z) \ \cup \ \{w'_i \mid b_i \in Z\}\right| - \left|Z \ \cup \ \{b'_i \mid b_i \in Z\}\right| \\ &= \left|\mathcal{N}_G(Z)\right| + \left|\{w'_i \mid b_i \in Z\}\right| - \left|Z\right| - \left|\{b'_i \mid b_i \in Z\}\right| \qquad (\clubsuit) \\ &= \left|\mathcal{N}_G(Z)\right| - \left|Z\right|. \end{aligned}$$

Now, consider an arbitrary set $Z' \subseteq B(G')$, set $Z = Z' \cap B(G)$, and observe that for any index $1 \leqslant i \leqslant n$

$$\begin{aligned} \left|\mathcal{N}_{G'}(Z')\right| - \left|Z'\right| &\geqslant \left|\mathcal{N}_{G'}\left(Z' \cup \{b'_i\}\right)\right| - \left|Z' \cup \{b'_i\}\right| \qquad \text{if } b'_i \notin Z', b_i \in Z', \\ \left|\mathcal{N}_{G'}(Z')\right| - \left|Z'\right| &= \left|\mathcal{N}_{G'}\left(Z' \setminus \{b'_i\}\right)\right| - \left|Z' \setminus \{b'_i\}\right| \qquad \text{if } b'_i \in Z', b_i \notin Z'. \end{aligned}$$

By repetitive application of the above relations we can arrive at Z'' such that $b'_i \in Z'' \iff b_i \in Z''$, i.e., $Z'' = Z \cup \{b'_i \mid b_i \in Z\}$. However, using equality $(\clubsuit)$ we have $\left|\mathcal{N}_{G'}(Z')\right| - \left|Z'\right| \geqslant \left|\mathcal{N}_{G'}(Z'')\right| - \left|Z''\right| = \left|\mathcal{N}_G(Z)\right| - \left|Z\right|$. Therefore, for any set $Y \subseteq B(G)$ that is a candidate for minimum surplus of X in G there exists set $Y' = Y \cup \{b'_i \mid b_i \in Y\}$ which is a corresponding candidate for minimum surplus of X in G'. Observe that similarly to equality $(\clubsuit)$ we have

$$\left|\mathcal{N}_{G'}(Y') \cup \left(X \cap W(G')\right)\right| - \left|Y'\right| = \left|\mathcal{N}_G(Y) \cup \left(X \cap W(G)\right)\right| - \left|Y\right|,$$

hence

$$\min \operatorname{surplus}_{G'}(X) \leqslant \min \operatorname{surplus}_G(X).$$

Case $(\geqslant)$**.** Let $Y' \subseteq B(G')$ be the set that maximizes $\left|Y' \setminus B(G)\right|$ among sets that realize the minimum surplus of X in G'. Suppose that for any $1 \leqslant t \leqslant n$ we have $b'_t \in Y'$ and set $Y = Y' \cap B(G)$. Note that due to equality $(\clubsuit)$ and the fact that Y' satisfies Hall's condition, any subset $Z \subseteq Y$ has at least $|Z|$ neighbors in G, that is, Y satisfies Hall's condition. Therefore, it is a valid candidate for the minimum surplus of X in G, and

$$\left|\mathcal{N}_{G'}(Y') \cup \left(X \cap W(G')\right)\right| - \left|Y'\right| = \left|\mathcal{N}_G(Y) \cup \left(X \cap W(G)\right)\right| - \left|Y\right|,$$

which implies $\min \operatorname{surplus}_{G'}(X) \geqslant \min \operatorname{surplus}_G(X)$.

Otherwise, assume there exists an index i such that $b'_i \notin Y'$. We denote by M a matching in G incremented by $\{b'_i, w'_i\}$, which saturates $X \cap B(G)$. It exists because $X \cap B(G)$ satisfies Hall's condition. For the same reason there is a Y'-saturating matching in G', which we denote by M'.

Let $\left[b'_i \overset{P}{\leftrightarrow} v\right]$ be the connected component of $M \cup M'$ that contains b'_i—due to $\deg(b'_i) = 1$ and Observation 1.1, it has to be a path starting in b'_i. Moreover, $v \in W(G')$ implies the existence of an M'-augmenting path that contradicts the fact that Y' realizes the minimum surplus of X in G'. Thus, we arrive at $v \in B(G')$. However, b'_i and w'_i are the only vertices of M that are outside of G, hence $v \in B(G)$. That means $v \notin X$, i.e., the set $Y'' = Y' \cup \{b'_i\} \setminus \{v\}$ is a valid candidate for minimum surplus of X in G'. Consider that at the same time Y'' realizes a value not bigger than Y' and $\left|Y'' \setminus B(G)\right| > \left|Y' \setminus B(G)\right|$, what is inconsistent with the definition of Y'. Therefore, $b'_t \in Y'$ for any $1 \leqslant t \leqslant n$. □

2.2 Trees

Trees constitute one of the simplest non-trivial cases for matching algorithms. Although, in the offline case there is an obvious greedy strategy that calculates maximum matching in $\mathcal{O}(|V|)$ time—as long as it is possible we repeatedly match an arbitrary leaf and remove the pair of vertices from the graph—in the online setting the logarithmic overhead is impossible to avoid in the worst case.

Consider a path consisting of 2^k black and 2^k white vertices, both sequences indexed from 0 to $n-1$. If we add the black vertices in ascending order of parity of their indices, that is,

$$1, 3, 5, 7, \ldots,\ 2, 6, 10, 14, \ldots,\ 4, 12, \ldots,\ 2^{k-1},\ 0,$$

then at each turn the algorithm will stumble upon a symmetric choice, both halves at the sides of b_t being isomorphic. Thus, in the worst case the total length of augmenting paths equals $(k+1) \cdot 2^k$. Figure 2.2 shows an example for $k = 3$ at several different turns; the red hue marks possible augmenting paths, while the numbers indicate the length of the one that

matched each particular vertex.

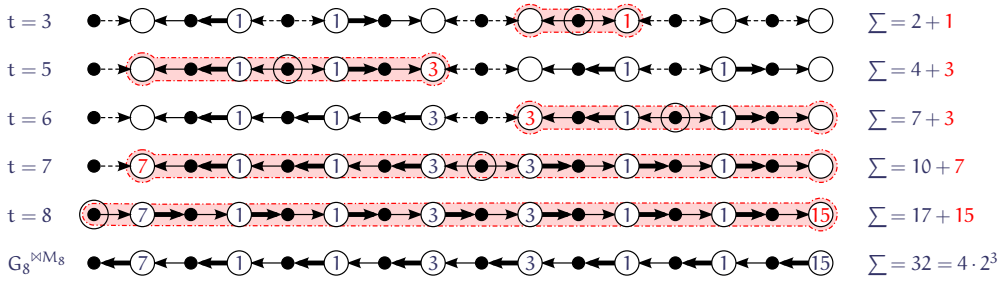

Figure 2.2: An example that achieves $\Omega(|V| \log |V|)$ worst-case lower bound on the total length of all augmenting paths for $k = 3$ at different turns. Each black vertex that arrives has exactly two indistinguishable augmenting paths to choose from—the one picked is indicated by a number that also denotes its length.

Hence, the aforementioned conjecture due to Chaudhuri, Daskalakis, Robert and Lin, states that the simple case of a path is asymptotically the worst possible. In this section we prove an intermediate claim which is stated precisely in Theorem 2.6. The proof of that result was first published at WAOA 2015 [19].

Theorem 2.6. *If G_n is a tree, then the total length of all the shortest augmenting paths is $\mathcal{O}(|V| \log^2 |V|)$.*

Note that we care only about the length of the augmenting paths, not about the sizes of search trees or overall complexity of the algorithm. In particular, because of Corollary 1.16, without loss of generality, we can assume that there is a perfect matching in G_n.

Before we start, we need to define a concept instrumental to the proof—the *dispatching vertex* of an augmenting path. It will allow us to split the augmenting paths into three cases, greatly simplifying the analysis.

Definition 2.7. Let $\left[b \overset{A}{\leftrightarrow} w\right]$ be an augmenting path going from $b \in B$ to $w \in W$. The *dispatching* vertex of A is the first vertex alive on that path, formally,

$$\operatorname{dis}(A) = \operatorname*{arg\,min}_{v \in V(A) \setminus \mathcal{D}} \operatorname{dist}_A(b, v).$$

Moreover, we split A by $\mathrm{dis}(A)$ into a prefix A^{pre} and suffix A^{dis}, that is

$$A = \left[b \xleftarrow{A^{\mathrm{pre}}} \mathrm{dis}(A)\right) \circ \left[\mathrm{dis}(A) \xleftarrow{A^{\mathrm{dis}}} w\right].$$

We refer to A^{dis} as the *dispatching path*. Note that, because of Lemma 1.11, the vertex $\mathrm{dis}(A)$, if it exists, is always black. In particular, it might be the very first vertex on A, in which case A^{pre} is empty. On the other hand, if there is no such vertex, we set $A^{\mathrm{pre}} = A$ and A^{dis} to be empty.

For any vertex $b \in B$ that was dispatching at some turn, the last turn in which b was the dispatching vertex is denoted by $\mathrm{tlast}(b)$ and the set of such turns as by $\mathcal{T}$, i.e.,

$$\mathrm{tlast}(b) = \min\{t \mid b = \mathrm{dis}(A_t)\}, \qquad \mathcal{T} = \{t \mid \exists b \in B.\ t = \mathrm{tlast}(b)\}. \quad \diamond$$

Observation 2.8. The vertices of A_t^{pre} die precisely at turn t, i.e.,

$$V(A_t^{\mathrm{pre}}) \subseteq \mathcal{D}_t \setminus \mathcal{D}_{t-1}.$$

Proof. The augmenting path A_t certifies that each vertex in $V(A_t^{\mathrm{pre}})$ was alive at turn $t-1$. However, because A_t^{pre} ends just before the first vertex alive in G_t, those are all dead at turn t. □

Lemma 2.9. *The sum of lengths of A_t^{pre} is linear in $|V|$,*

$$\sum_{t \in \{1,2,\ldots,n\}} |A_t^{\mathrm{pre}}| \in \mathcal{O}(|V|).$$

Proof. By Observation 2.8 each vertex of A_t^{pre} dies at turn t. Furthermore, because of Observation 1.12, any vertex can be in $V(A_t^{\mathrm{pre}})$ at most once, hence $\sum_{t \in \{1,2,\ldots,n\}} |A_t^{\mathrm{pre}}| \leqslant |V|$. □

Lemma 2.10. *The sum of lengths of non-final dispatching paths, i.e., A_t^{dis} such that $t < \mathrm{tlast}(\mathrm{dis}(A_t))$, is $\mathcal{O}(|V| \log |V|)$,*

$$\sum_{t \in \{1,2,\ldots,n\} \setminus \mathcal{T}} |A_t^{\mathrm{dis}}| \in \mathcal{O}(|V| \log |V|).$$

Proof. We first observe that every time some vertex $b \in B$ is dispatching not for the first time, one of its neighbours dies. To be more specific, if $b = \mathrm{dis}(A_t)$ and A_t does not start in b (what happens every but the first time b is dispatching), then $\{w, b\} \in E(A_t^{\mathrm{pre}})$ for some neighbour w of b. Based on Observation 2.8, the vertex w dies.

Hence, if b is a dispatching vertex for the k-th out of l times at turn t, then it has at least $l - k + 2$ alive white out-neighbours at t. Suppose that we discard two neighbors of b corresponding to the two heaviest subtrees, i.e., the two biggest connected components of $G_n \setminus \{b\}$ (see Section 1.1.2). Then for $k = l - 1$ we have at least one alive neighbor, for $k = l - 2$ we have at least two alive neighbors, that is, at least one alive neighbor other than the neighbor used at $k = l - 1$, and so on. In other words, for any $k < l$ we can find a distinct, not already assigned, alive neighbor w different than the two heaviest neighbors of b. However, the size of the subtree hanging in that neighbour bounds the length of the shortest augmenting path starting at b. Therefore, we can bound the total length of non-final paths dispatching at b by the total size of all but the two heaviest subtrees of b. Summing these over the whole tree gives us a $\mathcal{O}\big(|V| \log |V|\big)$ upper bound, as shown by the next lemma. □

Lemma 2.11. *Let T be an unrooted tree. For any vertex v let $S^v = \langle S_0^v, S_1^v, \ldots \rangle$ be the sequence of subtrees of v (i.e., the connected components of $T \setminus \{v\}$) ordered descending by their size, that is, $\big|V(S_i^v)\big| \geqslant \big|V(S_{i+1}^v)\big|$. Then for*

$$\Psi(v) = \sum_{i=2}^{|S^v|} \big|V(S_i^v)\big|$$

we have $\sum_{v \in V(T)} \Psi(v) \in \mathcal{O}\big(|T| \log |T|\big)$.

Proof. Let r be a centroid point of T, that is, a vertex such that $|V(S_0^r)| \leqslant \frac{1}{2}|V(T)|$. We root T at r, and perform the heavy-light decomposition of T (see Section 1.1.2). Observe that for all vertices $v \neq r$ we have that S_0^v contains r (it corresponds to the parent of v) and S_1^v corresponds to the biggest child of v. In other words, at most S_0^v and S_1^v can be connected by heavy edges, all the other subtrees $S_2^v, S_3^v, \ldots$ are connected by light edges.

Now we take an arbitrary vertex w and calculate how many times it can appear in $\sum_{v \in V(T)} \Psi(v)$. Suppose v is a vertex that counts w in $\Psi(v)$, then the first edge on the path from v to w has to be light, moreover, S_0^v is not counted in $\Psi(v)$, so that path cannot pass through the parent of v. Because of that v has to be an ancestor of w, however, there are at most $\mathcal{O}(\log|T|)$ light edges on any path from w to the root r for any w. In other words, there can be at most $\mathcal{O}(\log|T|)$ vertices that count w in its sum of Ψ. Summing that for all vertices of T we get the desired bound of $\mathcal{O}(|T|\log|T|)$. □

Lemma 2.12. *The sum of lengths of* A_t^{dis} *such that* $t = \text{tlast}(\text{dis}(A_t))$ *is bounded from above by* $\mathcal{O}(|V|\log^2|V|)$,

$$\sum_{t \in \mathcal{T}} \left|A_t^{\text{dis}}\right| \in \mathcal{O}(|V|\log^2|V|).$$

To prove the above lemma we will need a more fine-grained analysis than before. The problem with the shortest path approach is that the structure it gives us and the structure of matchings are much different. To close this gap we introduce yet another family of augmenting paths that relies much more on the structure of the tree. Obviously, because the shortest paths are shorter than any other path, any upper bound on the total length of the aforementioned new family of augmenting paths is an upper bound for the shortest paths as well.

Proof. Similarly to Lemma 2.11, we root G_n at a centeroid point and preform the heavy-light decomposition of the tree (see Section 1.1.2). Each vertex selects an edge to its biggest subtree, which we call heavy, all the other edges are light. The heavy edges form paths which happen to constitute a partition of $V(G_n)$. Denote by $\text{heavy path}(v)$ the heavy path to which v belongs. Formally $\text{heavy path}(v) \stackrel{\text{def}}{=} G_n\left[[v]_\sim\right]$ is the path induced by the equivalence class of v with regard to

$$v_1 \sim v_2 \stackrel{\text{def}}{\iff} \exists\left[v_1 \stackrel{P}{\leftrightarrow} v_2\right]. \min_{u \in V(P)} \text{level}(u) = \max_{u \in V(P)} \text{level}(u),$$

where $\mathrm{level} : V(G_n) \to \mathbb{N}$, which is the number of light edges on the simple path from a vertex to the root.

We define B_t as the M_{t-1}-augmenting path starting in b_t which is the greatest with respect to the lexicographic ordering on level's, that is,

$$\Big\langle \mathrm{level}(v_0),\ \mathrm{level}(v_1),\ \mathrm{level}(v_2),\ \ldots \Big\rangle$$

where $v_0, v_1, v_2, \ldots$ is the sequence of vertices of B_t. Note that all M_{t-1}-augmenting paths starting in b_t are the same until $\mathrm{dis}(A_t)$, in particular $\mathrm{dis}(B_t) = \mathrm{dis}(A_t)$ and $B_t^{\mathrm{pre}} = A_t^{\mathrm{pre}}$. To bound the length of B_t^{dis} we split it into B_t^{heavy} and B_t^{stree},

$$B_t^{\mathrm{heavy}} \stackrel{\mathrm{def}}{=} B_t^{\mathrm{dis}} \cap \mathrm{heavy\,path}\big(\mathrm{dis}(B_t)\big),$$
$$B_t^{\mathrm{stree}} \stackrel{\mathrm{def}}{=} B_t^{\mathrm{dis}} \setminus \mathrm{heavy\,path}\big(\mathrm{dis}(B_t)\big).$$

Observe that B_t^{dis} follows $\mathrm{heavy\,path}\big(\mathrm{dis}(B_t)\big)$ at most up to the closest vertex b such that $\mathrm{tlast}(b) > t$, namely

$$\big|B_t^{\mathrm{heavy}}\big| \leqslant \min\Big\{ \mathrm{dist}\big(b, \mathrm{dis}(B_t)\big) \;\Big|\; b \in B\Big(\mathrm{heavy\,path}\big(\mathrm{dis}(B_t)\big)\Big), \mathrm{tlast}(b) > t \Big\}.$$

This is because any vertex b with $\mathrm{tlast}(b) > t$ has at least three alive neighbors, with at least two of them reachable from b in ${G_t}^{\bowtie M_{t-1}}$, and at least one not on $\mathrm{heavy\,path}\big(\mathrm{dis}(B_t)\big)$. In turn, if b itself is not reachable from $\mathrm{dis}(B_t)$ in ${G_t}^{\bowtie M_{t-1}}$, then B_t^{heavy} must have ended earlier.

Consider an arbitrary heavy path H and the set D of dispatching vertices of H. Using $D_0 = D$ we split H into at least $d_0 = |D_0|$ non-empty fragments $h_0, h_1, \ldots, h_{d_0-1}$. Each such part h has either both its endpoints or at least one in D_0. Thus, we can assign h to one of its ending vertices, preferring earlier turn tlast, if there are two available. Formally $f_0 : \{h_i\}_i \to D_0$, where

$$f_0(h) = \operatorname*{arg\,min}_{b \,\in\, V(h) \cap D_0} \mathrm{tlast}(b).$$

Because of the inequality in the previous paragraph we have

$$\left|B^{\text{heavy}}_{\text{tlast}(f_0(h_i))}\right| \leqslant |h_i| \text{ for any } 0 \leqslant i < d_0.$$

This means that the length of H bounds the total length of B^{heavy}'s related to the dispatching vertices in the image of f_0. However, as at most two h_i's can be assigned to the same vertex of D_0, the image of f_0 constitutes at least a half of D_0.

To take care of the rest of D_0, we iterate this reasoning. We construct a sequence of sets $D_0 \supseteq D_1 \supseteq \ldots$, each step halving the size of D_i. More precisely, we set

$$D_i = D_{i-1} \setminus \left\{ f_{i-1}\left(h^{i-1}_j\right) \;\middle|\; 0 \leqslant j < d_{i-1} \right\},$$

where $h^i_0, h^i_1, \ldots, h^i_{d_i-1}$ are the parts of H after the split by D_i, while funcions $f_i : \{h^i_j\}_j \to D_i$ are defined as

$$f_i(h^i) = \operatorname*{arg\,min}_{b \,\in\, V(h^i) \cap D_i} \text{tlast}(b).$$

In other words, at most $\log|V|$ copies of H cover all B^{heavy} paths related to H. Summing that up over all the heavy paths gives us

$$\sum_{t \in \mathcal{T}} \left|B^{\text{heavy}}_t\right| \in \mathcal{O}\left(|V| \log |V|\right),$$

but it also means that for any $v \in V(H)$ at most $\log|V|$ of B^{stree} paths may start in v. More precisely, $\log|V|$ copies of all the non-heavy subtrees of $v \in V(H)$ cover all B^{stree} paths starting in v, which by Lemma 2.11, implies

$$\sum_{t \in \mathcal{T}} \left|B^{\text{stree}}_t\right| \in \mathcal{O}\left(|V| \log^2 |V|\right).$$

From the last two bounds we infer the statement of the Lemma 2.12. □

2.3 Playing Against an Adversary

We devote the last section of Chapter 2 to a quite surprising characteristic of Theorem 2.6 and its implications. Namely, nowhere in the proofs of Lemmas 2.9 to 2.12 we rely on the shape of any particular matching at any given turn, or even on the fact that these matchings are related to each other. To be more specific, we depend only on the structure of the tree, the properties of the dead and alive vertices, and, if we require maximality, the cardinality of the matchings. This leads us to a generalization of the setting in Definition 1.9 and a respective counterpart of Theorem 2.6.

We define the *adversarial dynamic augmenting path setting* as a setup similar to the one in Definition 1.9 with the difference that:

- there is an additional input sequence $M'_0, M'_1, \ldots, M'_n$ of matchings in $G_0, \ldots, G_n$ respectively, satisfying $|M'_i| = |M_i|$ for any $1 \leqslant i \leqslant n$;
- the algorithm outputs M'_{i-1}-augmenting paths and $M_i = M'_{i-1} \oplus A_i$.

Intuitively, each turn we are given a single black vertex with all its edges, however, the matching we use to calculate the shortest augmenting path is not the one produced by the algorithm in the previous turn, but some arbitrary matching of the same cardinality provided by the adversary. In particular, the edges might be oriented, wherever possible, away from the newly added vertex, thus making the augmenting paths the longest possible. Nonetheless, because we do not depend on the structure of the matching, the total length of all such augmenting paths is still small.

Corollary 2.13. If G_n in the above setting is a tree, then the total length of all the shortest augmenting paths is $\mathcal{O}(|V| \log^2 |V|)$. □

It seems that this is true also in general bipartite graphs, and thus we form the following conjecture.

Conjecture 2.14. *The total length of all the shortest augmenting paths in the setting above, that is, with the matching changing arbitrarily each turn, is still* $\mathcal{O}(|V| \log |V|)$ *worst case for any bipartite graph.*

The ramifications of that conjecture are twofold. First, it suggests a new perspective and a new research angle in which we are allowed to change the matching to fit into some schema. That could possibly lengthen the paths in the process, but it might make the problem a bit more predictable and less dynamic, hence, in some aspects, easier. Second, it might allow for better algorithms. A matching procedure based on the above idea could alter the calculated matching during some turns in a random way, thus perhaps making its worst case less bad. As the reasons behind this phenomenon are far from clear, in author's opinion Conjecture 2.14 is an interesting open problem.

Chapter 3

Ranks and Tiers

One significant drawback of the shortest augmenting path approach is that, finding such paths each turn takes linear time, but we are aiming for sublinear amortized efficiency. To solve this problem we introduce two notions, which will allow us to design an algorithm in Chapter 4 that solves dynamic matching problem Definition 1.9 in total running time of the offline maximum bipartite matching algorithm of Hopcroft and Karp [59].

These concepts are the *rank* and the *tier*. We start with pure mathematical definitions and then in Section 3.5 we relax them to allow for an efficient calculation.

Results included in this chapter appeared first at FOCS 2014 [18].

3.1 Definitions

Intuitively the rank measures how the algorithm, meaning searching procedures and augmenting paths, uses the graph. More precisely, it counts how many times we have visited a vertex or traversed an edge. In more operational terms, any object we would like to account for has a counter that is increased each time that object used.

The main idea is to minimize the maximum usage and thus spread the work of the algorithm as evenly as possible. Such an approach has several benefits. For example, one of the drawbacks of the shortest augmenting

path algorithm considered in the previous chapter is that it can produce in the graph uneven saturation with respect to the optimal matching M_n—some areas having an abundance of seeds, while other only very few of them. Such situation makes searching for seeds very costly, thus, it seems desirable to keep the graph in a more uniform state.

Naturally, dispersing the seeds more evenly may increase the total length of augmenting paths, in particular there could be fewer very short augmenting paths. Moreover, as we are not searching for the shortest augmenting paths anymore, we need another mechanism that could prevent us from repeatedly finding long augmenting paths. Fortunately, this is not an issue when employing the minimizing maximum usage strategy. Observe that long augmenting paths increase many more usage counters than short augmenting paths. Therefore, oversaturated areas will incur higher costs than other parts of the graph, and that will discourage the search procedure from entering during later turns. In other words bad decisions may happen, but they will not be repeated too often.

Finally, spreading the usage over the whole graph makes the complexity analysis in some ways easier. In the previous chapter to amortize the cost of applying the augmenting paths we had to devise a careful charging scheme that distributed the work of the algorithm globally, over different, sometimes only indirectly related parts of the graph. On the other hand, if the algorithm already does that by design, its costs can be amortized locally.

Throughout this thesis we use vertex-based rank, that is, a rank that measures the usage of vertices. One reason is that it simplifies the presentation, especially since we define rank only on white vertices, which are present since the beginning of the algorithm. The other reason is that it gives a slightly better bounds on the total length of augmenting paths.

Nonetheless, it is perfectly fine to define ranks on edges, in particular, the very first sketches of proofs were done in terms of edge-based ranks. For the sake of completeness, a brief consideration of edge-based ranks will be done in Section 3.4.

Definition 3.1. Let $G_0, G_1, \ldots$ and $M_0, M_1, \ldots$ be sequences of graphs and matchings as in Definition 1.9. We define $\mathrm{rank}_t(w)$ as the number of

augmenting paths up to turn t that visited w. Formally, $\mathrm{rank}_t : W(G_t) \to \mathbb{N}$ is given by

$$\begin{aligned}\mathrm{rank}_t(w) &\stackrel{\text{def}}{=} \begin{cases} \mathrm{rank}_{t-1}(w) + 1 & \text{if } w \in W(M_t \oplus M_{t-1}), \\ \mathrm{rank}_{t-1}(w) & \text{otherwise,} \end{cases} \\ &= \sum_{i=0}^{t} \mathbb{1}_{W(M_t \oplus M_{t-1})}(w).\end{aligned}$$

Although the domain of rank_t seem to depend on t, please recall that for any turn t we have $W(G_t) = W(G_0)$. We extend this definition to paths by taking the maximum of ranks of vertices

$$\mathrm{rank}_t(P) \stackrel{\text{def}}{=} \max_{w \in W(P)} \mathrm{rank}_t(w). \qquad \diamond$$

The second concept, *tier*, serves as a caching mechanism for the ranks. In the dynamic setting, each turn, when a black vertex b_t arrives, we try to match it with some seed, i.e., an unmatched white vertex (see Definition 1.10). To ensure good properties of the algorithm we try to do this using an augmenting path of the smallest possible rank. However, when searching for such a path, finding some low-rank vertex in the middle of the graph does not guarantee a low-rank passage—we might be forced to go through some high-rank vertices later. To solve this problem the tiers take into account all the minimum ranks that are necessary to reach a seed $s_t \in \mathcal{S}_t$.

Definition 3.2. We define $\mathrm{tier}_t : V(G_n) \to \mathbb{N} \cup \{\infty\}$ as the minimal rank over all paths in ${G_n}^{\bowtie M_t}$ leading to a seed

$$\mathrm{tier}_t(v) \stackrel{\text{def}}{=} \min\left\{ \mathrm{rank}_t(P) \;\middle|\; s \in \mathcal{S}_t,\ \left[v \xrightarrow{P} s\right] \in {G_n}^{\bowtie M_t} \right\},$$

where we assume a convention in which the minimum of an empty set is the positive infinity, that is, $\min \varnothing = \infty$. If a path $\left[v \xrightarrow{P} s\right]$ attains the minimum, namely $\mathrm{rank}_t(P) = \mathrm{tier}_t(v)$, then we say that P is a *witness* of $\mathrm{tier}_t(v)$. $\diamond$

Note that we could have defined tier_t in terms of ${G_t}^{\bowtie M_t}$ rather than ${G_n}^{\bowtie M_t}$, however, the current formula simplifies additions of new vertices, and M_t makes the additional black vertices irrelevant as shown by the next observation.

Observation 3.3. For any $v \in V(G_t)$ we have

$$\mathrm{tier}_t(v) = \min\Big\{\mathrm{rank}_t(P) \;\Big|\; s \in \mathcal{S}_t,\ \big[v \xrightarrow{P} s\big] \in {G_t}^{\bowtie M_t}\Big\}.$$

Proof. Any path in ${G_t}^{\bowtie M_t}$ is also a valid path in ${G_n}^{\bowtie M_t}$. Moreover, the set of white vertices does not change, $W(G_n) = W(G_0)$. Consider any $b \in B(G_n) \setminus B(G_t)$. As $b \notin B(M_t)$, all the edges incident to b in ${G_n}^{\bowtie M_t}$ are directed away from b. In other words, b cannot be reached from v and thus any path in ${G_n}^{\bowtie M_t}$ starting from v belongs to ${G_t}^{\bowtie M_t}$ as well. □

Although we define the tiers for both black and white vertices, the key role is played by the black vertices which are to pick the best white neighbor. In contrast, white vertices have at most one outgoing edge in ${G_t}^{\bowtie M_t}$ which has to be used by any path connecting it to a seed. However, despite being redundant, tiers of white vertices are still useful, in particular, they simplify formulas in Sections 3.2 and 3.3.

Observation 3.4. For any black vertex b we have

$$\mathrm{tier}_t(b) = \min\Big\{\mathrm{tier}_t(w) \;\Big|\; \langle b, w\rangle \in E\Big({G_n}^{\bowtie M_t}\Big)\Big\}.$$

For any white vertex w that is matched by M_t we have

$$\mathrm{tier}_t(w) = \max\big\{\mathrm{rank}_t(w), \mathrm{tier}_t\big(b_w^t\big)\big\}.$$ □

It is worth noting how the concept of augmenting paths of the minimum possible rank, relies on the assumption that the final graph G_n is bipartite. Certainly, the construction of the directed graph ${G_n}^{\bowtie M_t}$ requires G_n to be bipartite, yet one could imagine formulating the definition of tier in terms of just alternating paths. The issue here is that augmenting

paths have to be simple. Unfortunately, to ensure the compositionality of the tiers, so that their values have only local dependencies—ranks of their vertices and tiers of the closest neighbors—we need to be able to rearrange the witnesses by splitting them and then concatenating their parts. Such operations may produce non-simple paths, which would jeopardize their role as augmenting paths. However, the graph is bipartite and Observation 1.6 guarantees for any finite $\mathrm{tier}_t(v)$ the existence of a simple witness.

Yet, tiers provide even more structure. Their strength is that they provide exactly the information we need in any given turn to find an augmenting path of the minimum rank—it is enough just to follow non-increasing tiers. To substantiate these claims we introduce a few more notions.

Definition 3.5. We call an edge $e \in {G_t}^{\bowtie M_{t-1}}$ *tiered* if there exists a witness of $\mathrm{tier}_{t-1}(\mathrm{tail}(e)) < \infty$ that uses e. The set of all tiered edges of ${G_t}^{\bowtie M_{t-1}}$ is denoted by $E_t^\star$,

$$E_t^\star = \left\{ e \in E\left({G_t}^{\bowtie M_{t-1}}\right) \;\middle|\; \mathrm{tier}_{t-1}(\mathrm{head}(e)) \leqslant \mathrm{tier}_{t-1}(\mathrm{tail}(e)) < \infty \right\}.$$

Moreover, we call *tiered* any simple path P such that all its edges are tiered, i.e., $E(P) \subseteq E_t^\star$. Note that instead of t we use tiers in turn $t-1$, which represent the state of the graph before A_t is applied. ◇

Definition 3.6. A directed path $P \subseteq {G_t}^{\bowtie M_{t-1}}$ is *seeded* if it ends at any seed $s \in \mathcal{S}_t$. A *seeded tree* $T \subseteq {G_t}^{\bowtie M_{t-1}}$ is any tree rooted at some seed $s \in \mathcal{S}_t$ such that its every edge is oriented in the direction of the root s. A *seeded forest* $F \subseteq {G_t}^{\bowtie M_{t-1}}$ is a collection of $|\mathcal{S}_t|$ vertex-disjoint seeded trees that spans all the vertices of G_t that are able to reach some seed. ◇

Observe that not necessarily all the vertices are contained in some seeded tree—the algorithm may produce orientations in which some vertices cannot reach any seed. By Lemma 1.11 these vertices, that is, the vertices of infinite tier_{t-1}, are exactly the dead vertices $\mathcal{D}_{t-1}$ of G_{t-1} and potentially b_t if none of its neighbors were alive at $t-1$, i.e., $\mathcal{N}(b_t) \subseteq \mathcal{D}_{t-1}$. Thus, we may ignore these vertices, as no alternating path will visit them on any later turn. In particular, due to Corollary 1.16 it only makes sense

to consider the turns in which b_t can be matched, that is, b_t is of finite tier and belongs to some seeded tree. That allows us to construct a tiered seeded forest which spans all the vertices of finite tier_{t-1} and is the underlying reason for a number of characteristics of tiers that are the subject of the next section.

3.2 Basic Properties

Tiers give us a nice structure on ${G_t}^{\bowtie M_{t-1}}$. For example, a finite tier guarantees the existence of a path to some seed. In other words, to find such a path it is enough to follow the tiers down when possible, and we are bound to reach some seed.

Observation 3.7. For any turn t the following facts hold:

1. A vertex of $v \in V(G_n)$ is of finite tier_{t-1} if and only if $v \in G_{t-1}$ and it is alive in G_{t-1}, or $v \in B(G_n) \setminus B(G_{t-1})$ and it has a neighbor which is alive in G_{t-1}.

2. Any tiered seeded path is a witness for its starting vertex, that is, for any tiered seeded path $\left[v \xrightarrow{P} s\right] \in {G_n}^{\bowtie M_{t-1}}$ we have $\mathrm{rank}_{t-1}(P) = \mathrm{tier}_{t-1}(v)$.

3. For any $v \in V(G_n)$ of finite tier_{t-1}, there exists a simple tiered seeded path from v in $G_n\left[V(G_{t-1}) \cup \{v\}\right]^{\bowtie M_{t-1}}$, i.e., a simple tiered witness of $\mathrm{tier}_{t-1}(v)$.

4. There exists a tiered seeded forest in ${G_n}^{\bowtie M_{t-1}}$ that spans all the vertices of finite tier_{t-1}.

Proof. ***Fact 1.*** Certainly, due to Lemma 1.11, $v \in G_{t-1}$ is alive if and only if there is a path in ${G_{t-1}}^{\bowtie M_{t-1}}$ from v to some seed $s \in S_{t-1}$, which is also a valid path in ${G_t}^{\bowtie M_{t-1}}$. If $v \in B(G_n) \setminus B(G_{t-1})$, then all its neighbors are reachable in ${G_n}^{\bowtie M_{t-1}}$ and existence of such path is equivalent to existence of a white neighbor alive in G_{t-1}.

Fact 2. By the definition of tier, P is a candidate to be a witness, so $\mathrm{rank}_{t-1}(P) \geqslant \mathrm{tier}_{t-1}(v)$. However, P is tiered, so $\mathrm{rank}_{t-1}(P) \leqslant \mathrm{tier}_{t-1}(v)$ as well.

Fact 3. We start with $v_0 = v$ and pick a witness for $\mathrm{tier}_{t-1}(v_0)$ and follow it until we encounter the first vertex v_1 of strictly smaller tier $\mathrm{tier}_{t-1}(v_1) < \mathrm{tier}_{t-1}(v_0)$. Then we switch to the witness of $\mathrm{tier}_{t-1}(v_1)$ and follow it until we find a vertex v_2 of strictly smaller tier than v_1. As tiers are finite, this process has to stop, and the only way for it to stop is to reach a seed. The resulting path is a tiered seeded path, which can be made simple due to Observation 1.6.

Furthermore, all the vertices of $B(G_n) \setminus B(G_{t-1})$ have no incoming edges in ${G_n}^{\bowtie M_{t-1}}$, so the only vertex outside G_{t-1} can be v and the whole path fits into $G_n\left[V(G_{t-1}) \cup \{v\}\right]^{\bowtie M_{t-1}}$. Naturally, to find such a path algorithmically we could just use any searching method like the depth-first search on ${G_t}^{\bowtie M_{t-1}}$ constrained to $E_t^\star$.

Fact 4. Consider a subgraph $H \subseteq G_n$ which is the union of all the paths $\left[v \overset{P}{\leftrightarrow} s\right] \subseteq G_n$ such that $P^{\bowtie M_{t-1}} \subseteq {G_n}^{\bowtie M_{t-1}}$ is a shortest (simple) tiered witness for $\mathrm{tier}_{t-1}(v)$. Observe that $H^{\bowtie M_{t-1}}$ is a directed acyclic graph and that $V(H)$ are exactly the vertices of finite tier_{t-1}.

We construct $F \subseteq H$ by taking for each vertex v a single edge which is outgoing in $H^{\bowtie M_{t-1}}$ or nothing if $v \in S_{t-1}$. This makes $F^{\bowtie M_{t-1}}$ a collection of $|S_{t-1}|$ tiered seeded trees, that is, a tiered seeded forest that spans all the vertices of finite tier_{t-1}. □

We are now ready to prove one of the most important characteristics of tiers, that is, the monotonicity of tiers in t. To give a broader intuition we formalize it in two flavors, namely Lemma 3.8 and Lemma 3.9. The first one relies on the compositionality of tiers and how their values can depend only on the closest neighborhood—we utilize the structure of a seeded forest to explicitly maintain a graph-wide calculation order and derive the result inductively. While the proof of Lemma 3.8 involves almost the whole graph, the second lemma focuses instead on a subgraph that is

only directly related to the tier of vertex in question, namely its witness and the last augmenting path. It is worth reading because it uses very few additional concepts and thus clearly demonstrates the dependence between tiered augmenting paths and the monotonicity of tiers.

Lemma 3.8. *Suppose that the sequence of matchings $M_1, M_2, \ldots$ was constructed by applying each turn an augmenting path which was tiered. Then, for any vertex $v \in V(G_n)$ its tier $\mathrm{tier}_t(v)$ is a non-decreasing function of t.*

Proof. Fix some tiered seeded forest $F_{\bowtie}$ in ${G_n}^{\bowtie M_t}$ which exists by Observation 3.7.4. If vertex v is of infinite tier, trivially

$$\mathrm{tier}_{t-1}(v) \leqslant \infty = \mathrm{tier}_t(v),$$

so without loss of generality we can consider only vertices of finite tier, all of which belong to $V(F_{\bowtie})$. We proceed by induction on $\mathrm{dist}_{F_{\bowtie}}(v, \mathcal{S}_t)$.

Consider an arbitrary white vertex $w \in W(G_n)$ of finite tier. If w was a seed in turn $(t-1)$, then $\mathrm{tier}_{t-1}(w) = 0 \leqslant \mathrm{tier}_t(w)$. In particular, if $\mathrm{dist}_{F_{\bowtie}}(w, \mathcal{S}_t) = 0$, then w is a seed in turn t and by Corollary 1.14 it is also a seed in turn $(t-1)$.

On the other hand, if w is not a seed, nor it was in turn $(t-1)$, then

$$\begin{aligned}
\mathrm{tier}_t(w) &= \max\{\mathrm{rank}_t(w),\ \mathrm{tier}_t(b_w^t)\} \\
&\geqslant \max\{\mathrm{rank}_{t-1}(w),\ \mathrm{tier}_{t-1}(b_w^t)\} \\
&\geqslant \max\{\mathrm{rank}_{t-1}(w),\ \mathrm{tier}_{t-1}(b_w^{t-1})\} \\
&= \mathrm{tier}_{t-1}(w).
\end{aligned}$$

The first equality comes from the fact that $\langle w, b_w^t \rangle$ is the only edge outgoing from w in ${G_n}^{\bowtie M_t}$. That is also why $\langle w, b_w^t \rangle \in E(F_{\bowtie})$ and thus $\mathrm{dist}_{F_{\bowtie}}(w, \mathcal{S}_t) > \mathrm{dist}_{F_{\bowtie}}(b_w^t, \mathcal{S}_t)$. This allows us to use the inductive assumption on b_w^t, and as $\mathrm{rank}_t(w)$ is obviously non-decreasing in t the second line follows. The last line either is trivial if $b_w^t = b_w^{t-1}$, or is implied by the fact that the augmenting path in turn $(t-1)$, which had to go in direction $b_w^t \to w \to b_w^{t-1}$, was tiered, so $\mathrm{tier}_{t-1}(b_w^t) \geqslant \mathrm{tier}_{t-1}(b_w^{t-1})$.

Next, let $b \in B(G_n)$ be an arbitrary black vertex of finite tier, and set w to be its direct successor on the only tiered path in $F_{\bowtie}$ connecting b with a seed $s \in \mathcal{S}_t$. Then

$$\mathrm{tier}_t(b) = \mathrm{tier}_t(w) \geqslant \mathrm{tier}_{t-1}(w) \geqslant \mathrm{tier}_{t-1}(b),$$

where the first inequality depends on the inductive assumption, while the second inequality follows straight from Observation 3.4 if $\{b, w\} \notin M_{t-1}$, that is, $\langle b, w\rangle \in E\big({G_n}^{\bowtie M_{t-1}}\big)$, or, in case of $\{b, w\} \in M_{t-1}$, from the fact that the augmenting path which used $\langle w, b\rangle \in E\big({G_n}^{\bowtie M_{t-1}}\big)$ had to be tiered. □

Lemma 3.9. *Let* $\left[v \xrightarrow{P} s\right]$ *be the witness of* $\mathrm{tier}_t(v) = \mathrm{rank}_t(P) = r$, *then*

$$\forall v' \in V(P).\ \mathrm{tier}_{t-1}(v') \leqslant r.$$

Proof. Denote by $\left[b_{t-1} \xrightarrow{A} s_{t-1}\right]$ the tiered augmenting path in turn $t-1$. Let $\left(v_{A'} \xrightarrow{A'} s_{t-1}\right] \subseteq A$ and $\left(v_{P'} \xrightarrow{P'} s\right] \subseteq P$ be the longest suffixes of A and P such that respectively $\mathrm{rank}_{t-1}(A') \leqslant r$ and

$$\forall v' \in V(P').\ \mathrm{tier}_{t-1}(v') \leqslant r.$$

First, P' cannot be empty since s is a seed in turn t, so $\mathrm{tier}_{t-1}(s) = 0 \leqslant r$. Moreover, as tiered augmenting paths never go up-tier, for any $v' \in V(A) \setminus V(A')$ we have $\mathrm{tier}_{t-1}(v') > r$, so

$$\left(v_{P'} \xrightarrow{P'} s\right] \cap \left[b_{t-1} \xrightarrow{A \setminus A'} v_{A'}\right] = \varnothing,$$

that is, all the parts of A that P' intersects with are contained in a single seeded path of rank at most r, namely A'. Yet, observe that $A' \oplus P'$ contains a path from $v_{P'}$ to some seed—the graph is bipartite and we can construct a unit flow with sources at $v_{A'}$ and $v_{P'}$, and sinks in seeds s and s_{t-1}. Surely, $\mathrm{rank}_{t-1}\big(V(A') \cup V(P)\big) \leqslant r$, therefore, one of the paths in $A' \oplus P'$ constitutes a witness for $\mathrm{tier}_{t-1}(v_{P'}) \leqslant r$. However, since P' was defined to be the longest suffix, it implies that $v_{P'} = v$ and $P' = P$. □

Monotonicity of tiers in t has a number of consequences. Most importantly, it simplifies almost any argument that uses tiers and allows us to consider intuitively their values as a measure of complexity—how hard it is to find an augmenting path starting form some particular vertex, or following in some direction. Lemmas 3.8 and 3.9 are also crucial to the three, more concrete claims presented below.

Notably, the last one, Corollary 3.12, which claims that tiers are upper-bounded by the distance from the set of seeds, is the direct reason behind the main theorem of this chapter, Theorem 3.16.

Lemma 3.10. *For any edge $e \in G_n{}^{\bowtie M_t}$ we have*

$$\begin{aligned} \operatorname{rank}_t(\operatorname{tail}(e)) &\leqslant \operatorname{tier}_t(\operatorname{head}(e)) + 1 && \text{if } e \text{ is matched in } t, \\ \operatorname{tier}_t(\operatorname{tail}(e)) &\leqslant \operatorname{tier}_t(\operatorname{head}(e)) && \text{otherwise.} \end{aligned}$$

Proof. If e is matched in t then $\operatorname{tail}(e) = w \in W(G_n)$ is a white vertex such that $\operatorname{rank}_t(w) > 0$ and $\operatorname{head}(e) = b_w^t$ is a black vertex. Let τ be the last time the rank of w changed, i.e.,

$$\tau = \max\left\{ i \in \{0, 1, \ldots, t-1\} \;\middle|\; \operatorname{rank}_{i-1}(w) < \operatorname{rank}_i(w) \right\}.$$

Then, since the augmenting path A_τ was tiered, it holds that $\operatorname{tier}_{\tau-1}(b_w^t) \geqslant \operatorname{rank}_{\tau-1}(w)$. However, by Lemma 3.8, the tier does not decrease, so

$$\operatorname{tier}_t(b_w^t) + 1 \geqslant \operatorname{tier}_{\tau-1}(b_w^t) + 1 \geqslant \operatorname{rank}_{\tau-1}(w) + 1 = \operatorname{rank}_t(w).$$

On the other hand, if e is not matched in t, then $\operatorname{tail}(e)$ is black and the inequality follows directly from Observation 3.4. □

Corollary 3.11. Let P be a seeded path, then every white vertex $w \in W(P)$ of rank $\operatorname{rank}_t(w) \geqslant 1$ is followed by some vertex $w' \in W(P)$ of rank $\operatorname{rank}_t(w') \geqslant \operatorname{rank}_t(w) - 1$.

Proof. By Lemma 3.10 $\operatorname{tier}_t(b_w) \geqslant \operatorname{rank}_t(w) - 1$. By definition of tier, any path from w to any seed, including P, contains a white vertex w' such that $\operatorname{rank}_t(w') \geqslant \operatorname{tier}_t(w)$. □

Corollary 3.12. The length of the shortest directed path from $v \in V(G_n)$ to a seed in ${G_n}^{\bowtie M_{t-1}}$ is at least $2\,\mathrm{tier}_{t-1}(b)$. □

3.3 Upper Bounds on Ranks and Tiers

In this section we prove the main results of this chapter, which bound from above the ranks and tiers generated by any algorithm that maintains its matching using tiered augmenting paths.

The direct reason behind Theorem 3.16 is Corollary 3.12 which establishes that tiers, and therefore also ranks, are bounded from above by the distance to the set of seeds. This relationship allows us to formulate the following corollary.

Corollary 3.13. For any t and a non-empty set $\mathcal{P}$ of vertex-disjoint seeded paths of finite rank in ${G_n}^{\bowtie M_t}$

$$\min\{\mathrm{rank}_t(P) \mid P \in \mathcal{P}\} \leqslant \frac{|M_t \cap E(\mathcal{P})|}{|\mathcal{P}|}.$$

Proof. If there exists a path of $\mathrm{rank}_t(P) = 0$, then the inequality is trivial. Otherwise, in each path we have a white vertex of rank at least one, and by Corollaries 3.11 and 3.12 each such path has to have at least $\mathrm{rank}_t(P)$ matched edges. □

The general idea is that a single vertex of rank r implies the existence of $\lfloor r/2 \rfloor$ disjoint paths, each of rank at least $\lceil r/2 \rceil$. However, because of Corollary 3.13 we have then that $\lceil r/2 \rceil \leqslant |M_n| \cdot \lfloor r/2 \rfloor^{-1}$, which implies $r \in \mathcal{O}(|V|^{1/2})$.

To this end we introduce Lemma 3.14 which captures a certain reversed behavior of a tiered algorithm crucial for our proof, pictured in Figure 3.1. The diagram presents the state of the graph after (to the left: graph ${G_t}^{\bowtie M_t}$) and before (to the right: graph ${G_t}^{\bowtie M_{t-1}}$) the augmentation step. Assume that graph ${G_t}^{\bowtie M_t}$ contains seeded paths $Q_1 \dots Q_l$. On the picture to the left, the ranks of these paths are shown by the heights of the corresponding

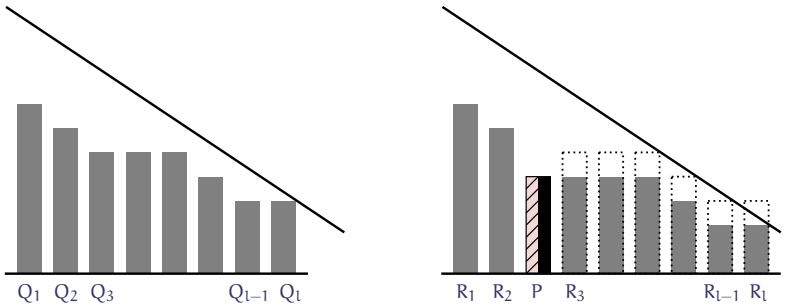

Figure 3.1: The bars corresponding to the paths after (to the left: graph $G_t^{\bowtie M_t}$) and before (to the right: graph $G_t^{\bowtie M_{t-1}}$) the augmentation step.

bars, sorted by rank. Lemma 3.14 states, that before the augmentation step (the right side of the picture), each path Q_i had its counterpart R_i, plus there was one more path P. 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 P) dominates the bars that decreased.

Hence, the line inclined at the hypothetical angle of 45 degrees (or more precisely at a single rank unit per bar width) through the right top point of the bar representing R_l does not drop as we go from t to $t-1$ backwards in time—whenever the bars reduce their height by one, an additional bar appears to support the line at its initial position.

The idea 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 disjoint paths whose height is also linear in r.

It is worth mentioning that in the following reasoning it is enough to consider vertices of finite rank. Other vertices affect neither the ranks nor the finite tiers.

Lemma 3.14. *Let $Q_1, Q_2, \ldots, Q_l$ be a sequence of vertex-disjoint seeded paths in $G_t^{\bowtie M_t}$. Also, let $b_t \xrightarrow{A_t} s$ be the tiered augmenting path in turn t, i.e., in $G_t^{\bowtie M^{t-1}}$. Then, there exist vertex-disjoint seeded paths $R_1, \ldots, R_l$ and P in*

$G_{t-1}{}^{\bowtie M_{t-1}}$ *such that*

$$\begin{aligned}
&\mathrm{rank}_{t-1}(A_t) \leqslant \mathrm{rank}_{t-1}(P),\\
&\quad \mathrm{rank}_t(Q_i) \leqslant \mathrm{rank}_{t-1}(R_i) && \textit{for } \mathrm{rank}_t(Q_i) > \mathrm{rank}_{t-1}(A_t) + 1,\\
&\quad \mathrm{rank}_t(Q_i) \leqslant \mathrm{rank}_{t-1}(R_i) + 1 && \textit{otherwise.}
\end{aligned}$$

Proof. Let w_i be the highest-ranked white vertex of Q_i that is the closest to the seed. Without loss of generality we can assume that Q_i starts with w_i. Then, for $i \in \{1, 2, \ldots, l\}$, directed path Q_i is of the form $\left[w_i \xrightarrow{Q_i} s_i\right]$ where $s_i \in \mathcal{S}_t$ is a seed.

Now, we define a $0/1$-flow network F by

$$\begin{aligned}
V(F) &= V(G_t) \cup \{\sigma, \tau\}\\
E(F) &= E\left(G_t{}^{\bowtie M_{t-1}}\right)\\
&\quad \cup \left\{\langle \sigma, b_t\rangle, \langle \sigma, w_1\rangle, \ldots, \langle \sigma, w_l\rangle\right\}\\
&\quad \cup \left\{\langle s, \tau\rangle, \langle s_1, \tau\rangle, \ldots, \langle s_l, \tau\rangle\right\}
\end{aligned}$$

where σ and τ are artificially added source and sink respectively (see Figure 3.2).

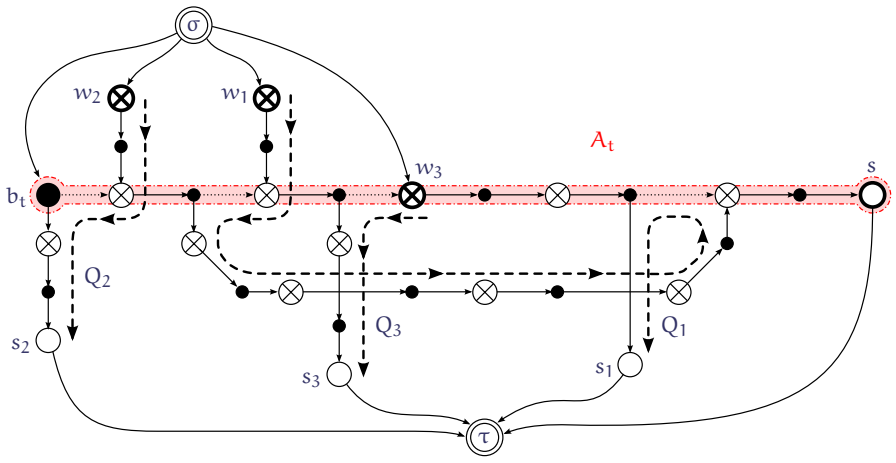

Figure 3.2: An example of a flow network F, where solid arcs form the flow from source σ to sink τ with the total value 4.

Because $\left[b_t \xrightarrow{A_t} s\right]$ is an augmenting path expanding matching in graph ${G_t}^{\bowtie M_{t-1}}$, we have that $\left[\sigma \to b_t \xrightarrow{A_t} s \to \tau\right]$ 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 ${G_t}^{\bowtie M_t}$. Because $Q_1, \ldots, Q_l$ are pairwise vertex-disjoint directed paths in ${G_t}^{\bowtie M_t}$, we have that paths Q'_i of the form $\left[\sigma \to w_i \xrightarrow{Q'_i} s_i \to \tau\right]$ are pairwise edge-disjoint directed paths in residual graph F^{f_1}. Consecutive application of the augmenting paths $Q'_1, \ldots, Q'_l$ 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 σ to τ as f_{l+1} is a $0/1$-flow. Because $b_t, w_1, \ldots, w_l$ are all neighbors of σ and $s_1, \ldots, s_l, s_{l+1} = s$ are all neighbors of τ, we obtain $(l+1)$ paths of the form

$$\left[\sigma \to b_t \to w_{l+1} \xrightarrow{P} s_{\aleph(l+1)} \to \tau\right] \text{ and } \left[\sigma \to w_i \xrightarrow{R_i} s_{\aleph(i)} \to \tau\right]$$

for $i \in \{1, 2, \ldots, l\}$, where $\aleph$ is some permutation of $\{1, 2, \ldots, l+1\}$ and $R_1, \ldots, R_l, P$ are edge-disjoint directed paths in ${G_t}^{\bowtie M_{t-1}} \subseteq F$. Because $w_1, \ldots, w_l, b_t$ are pairwise different and so are $s_1, \ldots, s_l, s_{l+1}$, moreover, each vertex in G_{t-1} has out-degree or in-degree equals at most 1, we have that paths $R_1, \ldots, R_l, P$ are also vertex-disjoint.

To see the first inequality of Lemma 3.14, it is enough to mention that path A_t was chosen in turn t so it was the smallest rank seeded path from b_t. Because the augmenting path A_t in turn t uses white vertices with rank at most $\mathrm{rank}_{t-1}(A_t)$, the two other inequalities are obvious. □

The next lemma, together with Corollary 3.13 imply the core of our result. Imagine that for some white vertex w its rank was raised to $\mathrm{rank}_\tau(w)$ by some turn τ. The lemma states that for each $j \in \{1, 2, \ldots, \mathrm{rank}_\tau(w)\}$ we can find a turn $t \leqslant \tau$ and a collection $\mathcal{B}_t$ of j vertex-disjoint paths in ${G_t}^{\bowtie M_t}$ such that the minimum of their ranks was at least $\mathrm{rank}_\tau(w) - |\mathcal{B}|$. The proof heavily bases on Lemma 3.14, which allows us, in a sense, to reverse the steps of the algorithm.

We begin the reversing process in turn t', where a path P exists with rank at least $\mathrm{rank}_\tau(w) - 1$. Starting from a one element collection $\{P\}$, we iteratively construct larger collections of paths using Lemma 3.14—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 increases by one
- there is a moment t_0 when all ranks in the collection are equal to 0.

Let us now move on to formalizing this idea.

Lemma 3.15. *For any turn $\tau \in \{1, 2, \ldots, n\}$, white vertex $w \in W_\tau$ and natural number $j \in \{1, 2, \ldots, \mathrm{rank}_\tau(w)\}$ there is turn $t < \tau$ and a collection $\mathcal{B}_t$ of j pairwise vertex-disjoint seeded simple paths $\mathcal{B}_t = \{\mathcal{K}_1, \ldots, \mathcal{K}_j\}$ in ${G_t}^{\bowtie M_t}$ such that*

$$\forall \mathcal{K} \in \mathcal{B}_t.\ \mathrm{rank}_t(\mathcal{K}) \geqslant \mathrm{rank}_\tau(w) - |\mathcal{B}_t|.$$

Proof. Let $t' \leqslant \tau$ be the round when the rank of white vertex w was changed the last time (weakly) before round τ. Now, we define sequence $\mathcal{B}_{t'-1}, \ldots, \mathcal{B}_0$, where $\mathcal{B}_k$ is a collection of vertex-disjoint simple seeded paths in ${G_k}^{\bowtie M_k}$ for $k \in \{0, 1, \ldots, t'-1\}$. From the definition of t' we know that $\mathrm{rank}_\tau(w) = \mathrm{rank}_{t'}(w)$ and $\mathrm{rank}_{t'-1}(A_{t'}) \geqslant \mathrm{rank}_{t'-1}(w)$. If we remove the vertex $b_{t'}$ from $A_{t'}$, we obtain a simple path P in ${G_{t'-1}}^{\bowtie M_{t'-1}}$ with rank

$$\begin{aligned}\mathrm{rank}_{t'-1}(P) &= \mathrm{rank}_{t'-1}(A_{t'}) \\ &\geqslant \mathrm{rank}_{t'-1}(w) \\ &= \mathrm{rank}_{t'}(w) - 1 = \mathrm{rank}_\tau(w) - 1.\end{aligned}$$

Thus, we define $\mathcal{B}_{t'-1} = \{P\}$. Now, let us assume that $\mathcal{B}_k = \{Q_1, \ldots, Q_l\}$ is defined for some $k \in \{1, 2, \ldots, t'-1\}$. By applying Lemma 3.14 to a sequence of paths $Q_1, \ldots, Q_l \subseteq {G_k}^{\bowtie M_k}$ and augmenting path A_k, we obtain

simple seeded paths $R_1, \ldots, R_l, \overline{P}$ in ${G_{k-1}}^{\bowtie M_{k-1}}$ satisfying the inequalities of the lemma. Then we define $\mathcal{B}_{k-1}$ as

$$\mathcal{B}_{k-1} = \begin{cases} \{R_1, \ldots, R_l, \overline{P}\} & \exists i \in \{1, 2, \ldots, l\}.\ \mathrm{rank}_k(Q_i) > \mathrm{rank}_{k-1}(R_i), \\ \{R_1, \ldots, R_l\} & \forall i \in \{1, 2, \ldots, l\}.\ \mathrm{rank}_k(Q_i) = \mathrm{rank}_{k-1}(R_i). \end{cases}$$

If $\mathrm{rank}_k(Q_i) > \mathrm{rank}_{k-1}(R_i)$ for some $i \in \{1, 2, \ldots, l\}$ then

$$\mathrm{rank}_k(Q_i) \leqslant \mathrm{rank}_{k-1}(A_k) + 1 \leqslant \mathrm{rank}_{k-1}(\overline{P}) + 1,$$

by inequalities of Lemma 3.14. Applying them one more time yields

$$\min_{R \in \mathcal{B}_{k-1}} \mathrm{rank}_{k-1}(R) \geqslant \begin{cases} \min\limits_{Q \in \mathcal{B}_k} \mathrm{rank}_k(Q) - 1 & \text{if } |\mathcal{B}_{k-1}| > |\mathcal{B}_k| \\ \min\limits_{Q \in \mathcal{B}_k} \mathrm{rank}_k(Q) & \text{if } |\mathcal{B}_{k-1}| = |\mathcal{B}_k| \end{cases}$$

Repeating these steps multiple times gives us

$$\min_{R \in \mathcal{B}_k} \mathrm{rank}_k(R) \geqslant \min_{Q \in \mathcal{B}_{t'-1}} \mathrm{rank}_{t'-1}(Q) - \left(|\mathcal{B}_k| - 1\right) = \mathrm{rank}_\tau(w) - |\mathcal{B}_k|,$$

for any chosen $k \in \{0, 1, \ldots, t' - 1\}$. The above formula implies that family $\mathcal{B}_0$ is of size at least $\mathrm{rank}_\tau(w)$ due to $\mathrm{rank}_0(R) = 0$ for all $R \in \mathcal{B}_0$. As the size of collections $(\mathcal{B}_k)_k$ differs between turns at most by one, we have that for any $j \in \{1, 2, \ldots, \mathrm{rank}_\tau(w)\}$ there is some $k \in \{0, 1, \ldots, t' - 1\}$ such that $|B_k| = j$ which concludes the proof. □

We are now ready to prove the main theorem.

Theorem 3.16. *For any dynamic unweighted matching algorithm that uses tiered augmenting paths it holds that* $\mathrm{rank}_n(w) \leqslant 2|M_n|^{1/2}$ *for every* $w \in W$.

Proof. By the previous lemma there is a time $t \in \{0, 1, \ldots, n - 1\}$ where there are sufficiently many paths of sufficiently high ranks, as illustrated in Figure 3.3. Again, each bar represents a path and the heights of the bars are the ranks of the corresponding paths. Thus, we can choose $t \in$

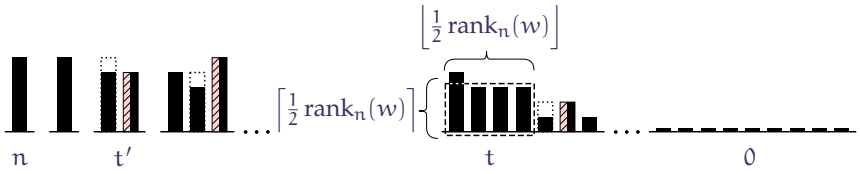

Figure 3.3: In time t there is a collection of $\lfloor \operatorname{rank}_n(w)/2 \rfloor$ paths of rank at least $\lceil \operatorname{rank}_n(w)/2 \rceil$. The mixed-color bar is the one that disappears at that turn, i.e., is considered responsible for causing the other bars to grow, which was marked with a dotted outline.

$\{1, 2, \ldots, t'\}$ for which $|\mathcal{B}_t| = \lfloor \frac{1}{2} \operatorname{rank}_n(w) \rfloor$, thus

$$\min_{R \in \mathcal{B}_t} \operatorname{rank}_t(R) \geqslant \left\lceil \frac{1}{2} \operatorname{rank}_n(w) \right\rceil.$$

However, by applying Corollary 3.13 we get

$$\left\lceil \frac{1}{2} \operatorname{rank}_n(w) \right\rceil \leqslant \min_{R \in \mathcal{B}_t} \operatorname{rank}_t(R) \leqslant \frac{|M_n|}{|\mathcal{B}_t|} = \frac{|M_n|}{\lfloor \frac{1}{2} \operatorname{rank}_n(w) \rfloor}$$

which can be transformed into

$$\left\lfloor \frac{1}{4} \operatorname{rank}_n(w)^2 \right\rfloor \leqslant \left\lceil \frac{1}{2} \operatorname{rank}_n(w) \right\rceil \cdot \left\lfloor \frac{1}{2} \operatorname{rank}_n(w) \right\rfloor \leqslant |M_n|,$$

that is, $\operatorname{rank}_n(w) \leqslant 2|M_n|^{1/2}$. □

Corollary 3.17. For any $v \in V(G_n)$ we have $\operatorname{tier}_n(v) \leqslant 2|M_n|^{1/2}$. □

3.4 Focusing on the Edges

Sometimes it is advantageous to focus on edges, rather than vertices. For example, consider the augmenting path approach in the setting of unit flows. Both ranks and tiers could be defined in the same way and many of the results obtained in this chapter would still be true. Unfortunately, Corollary 3.13 requires the paths in collection $\mathcal{P}$ to be vertex-disjoint, but that is not possible anymore, in particular, Lemma 3.14 is not valid. Yet,

we could weaken that assumption and consider what would happen if these paths were edge-disjoint.

Luckily, ranks and tiers work well in this case. As only slight changes are necessary to translate the original arguments of Section 3.3 to the new setting, the claims in this section are given mostly without proofs. We start with two versions of Corollary 3.13.

Corollary 3.18. For any t and a non-empty set $\mathcal{P}$ of edge-disjoint seeded paths of finite rank in ${G_n}^{\bowtie M_t}$

$$\min\{\mathrm{rank}_t(P) \mid P \in \mathcal{P}\} \leqslant \frac{|E(\mathcal{P})|}{|\mathcal{P}|}.$$ □

Corollary 3.19. For any t and a non-empty set $\mathcal{P}$ of edge-disjoint seeded paths of finite rank in ${G_n}^{\bowtie M_t}$

$$\min\{\mathrm{rank}_t(P) \mid P \in \mathcal{P}\} \leqslant \frac{|V(\mathcal{P})|}{|\mathcal{P}|^{1/2}}.$$

Proof. Set $r = \min\{\mathrm{rank}_t(P) \mid P \in \mathcal{P}\}$ and let c_i be the number of vertices of tier i in $V(\mathcal{P})$,

$$c_i = \left|\{v \in V(\mathcal{P}) \mid \mathrm{tier}_t(v) = i\}\right|.$$

However, because there can be at most $c_i \cdot c_{i+1}$ edges between levels i and $i+1$, we can bound the size of $\mathcal{P}$ using the square of the average number of vertices per level:

$$\begin{aligned}|\mathcal{P}| &\leqslant \min\{c_i \cdot c_{i+1} \mid 0 \leqslant i < r\} \\ &\leqslant \left(\min\{\tfrac{1}{2} \cdot (c_i + c_{i+1}) \mid 0 \leqslant i < r\}\right)^2 \leqslant \left(\frac{|V(\mathcal{P})|}{r}\right)^2.\end{aligned}$$ □

Theorem 3.20. *For any algorithm that uses tiered augmenting paths which can be made edge-disjoint it holds for every $w \in W$ that*

$$\mathrm{rank}_n(w) \in \mathcal{O}\left(\min\left\{|E(G)|^{1/2},\ |V(G)|^{2/3}\right\}\right).$$ □

Corollary 3.21. For any $v \in V(G)$ for the above algorithm we have

$$\mathrm{tier}_n(v) \in \mathcal{O}\Big(\min\Big\{\big|E(G)\big|^{1/2}, \big|V(G)\big|^{2/3}\Big\}\Big). \qquad \square$$

It is possible to take this a step further, that is, define the ranks in terms of edges. The bounds this approach yields are not as good, but the proofs are a bit simpler. Observe, that we still define tiers in terms of vertices, only using the edge-based ranks instead.

Definition 3.22. Set $\mathrm{rank}_t^E : E(G_t) \to \mathbb{N}$ and $\mathrm{tier}_t^E : V(G_n) \to \mathbb{N} \cup \{\infty\}$ as

$$\begin{aligned}
\mathrm{rank}_t^E(e) &\stackrel{\text{def}}{=} \sum_{i=0}^{t} \mathbb{1}_{M_t \oplus M_{t-1}}(e) \\
&= \begin{cases} \mathrm{rank}_{t-1}^E(w) + 1 & \text{if } e \in M_t \oplus M_{t-1}, \\ \mathrm{rank}_{t-1}^E(w) & \text{otherwise,} \end{cases} \\
\mathrm{rank}_t^E(P) &\stackrel{\text{def}}{=} \max_{e \in E(P)} \mathrm{rank}_t^E(w), \\
\mathrm{tier}_t^E(v) &\stackrel{\text{def}}{=} \min\Big\{\mathrm{rank}_t^E(P) \;\Big|\; s \in \mathcal{S}_t,\ \big[v \xrightarrow{P} s\big] \in G_n{}^{\bowtie M_t}\Big\}.
\end{aligned} \qquad \diamond$$

Lemma 3.23. $\mathrm{tier}_t^E(v)$ *is non-decreasing in* t *for any* $v \in V(G)$. $\square$

Unsurprisingly, edge-based ranks have their own counterpart of Theorem 3.16. Although the asymptotic bound is of the same order, the fact that the rank is defined on edges makes a significant difference. In particular, as we will see in the next chapter, Theorems 3.16 and 3.36 imply that the total length of all the augmenting paths is $\mathcal{O}\big(|V|^{3/2}\big)$, while the theorem below would give us only $\mathcal{O}\big(|E| \cdot |V|^{1/2}\big)$.

Theorem 3.24. *For any dynamic unweighted matching algorithm that uses tiered augmenting paths it holds that* $\mathrm{rank}_n^E(e) \leqslant 2\big|V(G_n)\big|^{1/2}$ *for every edge* $e \in E(G_n)$. $\square$

Corollary 3.25. For any $v \in V(G_n)$ we have $\mathrm{tier}^E_n(v) \leqslant 2\left|V(G_n)\right|^{1/2}$. □

Finally, similarly for other, non-matching algorithms for which their intrinsic augmenting paths can be made edge-disjoint, we have a counterpart of Theorem 3.20.

Theorem 3.26. *For any algorithm that uses tiered augmenting paths which can be made edge-disjoint it holds for every edge $e \in E(G_n)$ that*

$$\mathrm{rank}^E_n(e) \in \mathcal{O}\left(\min\left\{\left|E(G_n)\right|^{1/2}, \left|V(G_n)\right|^{2/3}\right\}\right).$$ □

Corollary 3.27. For any $v \in V(G_n)$ for the above algorithm we have

$$\mathrm{tier}^E_n(v) \in \mathcal{O}\left(\min\left\{\left|E(G_n)\right|^{1/2}, \left|V(G_n)\right|^{2/3}\right\}\right).$$ □

3.5 Relaxation of Ranks and Tiers

The ranks and tiers as defined in Definitions 3.1 and 3.2 admit some interesting properties, however, they are not easy to apply in an algorithmic setting. Although we could use them to find augmenting paths of total length bounded from above by $\mathcal{O}(|V|^{3/2})$, it is far from straightforward how to do it in an efficient manner.

One crucial drawback is that it is not clear how to keep the tiers updated from turn to turn—not only the dependency graph may have cycles, but can it change significantly every round. Moreover, even if managing tiers could be done efficiently, it would immensely complicate the procedure. Another disadvantage is that ranks and tiers focus solely on the cost of applying augmenting paths, yet each such path has first to be found. However, it is usually the searching phase that poses the greatest challenge and determines the final running time of the algorithm.

Fortunately, there is a simple solution—instead of managing strict ranks and tiers defined by Definitions 3.1 and 3.2, we relax posed requirements. Most importantly, this allows us to use the ranks to account

for the work done during the searching phase. Furthermore, it gives the algorithm more freedom, in particular, we will be able to update the tiers while searching the graph.

To this end, to accommodate the search, we allow additional optional increases of the rank when the augmenting path could have visited the vertex, instead of just when it did. Of course, we need the theorems related to the old ranks and tiers to apply to relaxed ranks and corresponding tiers. Luckily they do, given the augmenting paths we use are tiered according to ranks that satisfy the four properties shown below—properties which we have implicitly used in the proofs presented in previous sections.

Definition 3.28. Let $\mathrm{rank}^\star$ be a sequence of rank functions, that is, $\mathrm{rank}^\star_t : W(G_n) \to \mathbb{N}$ for $1 \leqslant t \leqslant n$, together with the corresponding tiers:

$$\mathrm{tier}^\star_t(v) \stackrel{\text{def}}{=} \min\Big\{\mathrm{rank}^\star_t(P) \;\Big|\; s \in S_t,\ \big[v \xrightarrow{P} s\big] \in {G_t}^{\bowtie M_t}\Big\}.$$

We say that $\mathrm{rank}^\star$ is a *relaxed rank*, and $\mathrm{tier}^\star$ is a *relaxed tier* if they together satisfy the following properties:

1. $\mathrm{rank}^\star_t(w)$ is non-decreasing in t,
2. $\mathrm{rank}^\star_t(w)$ increases when an augmenting path uses w,
3. $\mathrm{rank}^\star_t(w) \leqslant \max\big\{\mathrm{tier}^\star_{t-1}(b^t_w) + 1,\ \mathrm{tier}^\star_{t-1}(w)\big\}$,
4. $\mathrm{rank}^\star_t(w) \leqslant \max\big\{\mathrm{rank}^\star_{t-1}(A_t) + 1,\ \mathrm{tier}^\star_{t-1}(w)\big\}$. ◇

The first two properties are necessary if we would like to use ranks and tiers to bound the complexity of the algorithms from Chapter 4. Although Property 3 could be a bit weaker for Lemma 3.10 (see the proof of Lemma 3.30), we still need it in this form, along with Property 4, in the proof of Lemma 3.14 (see the corresponding Lemma 3.34). The two inequalities ensure respectively that the rank of any path Q_i from Lemma 3.14 can grow with respect to R_i at most by one per turn, and that only paths which ranks were smaller or equal to the rank of the augmenting path may grow at all. These two properties are essential to Lemma 3.14, otherwise the resulting collection of paths might not satisfy

the inequalities that are later necessary in Lemma 3.15. A more detailed sketch of the argument is given below, after the statement of Lemma 3.34.

Properties 3 and 4 hold for the original rank_t, because we increase the ranks each time at most by one, and only when the augmenting path uses the related vertex. More precisely, if the augmenting path does not use w, then $\mathrm{rank}_t(w) = \mathrm{rank}_{t-1}(w) \leqslant \mathrm{tier}_{t-1}(w)$. On the other hand, if $w \in W(A_t)$, then because A_t is tiered $\mathrm{rank}_{t-1}(w) = \mathrm{tier}_{t-1}(b^t_w)$ and $\mathrm{rank}_{t-1}(w) \leqslant \mathrm{rank}_{t-1}(A_t)$.

Since the relaxed ranks, in contrast to the strict ranks, can be increased during the algorithm run in a number of circumstances, in the context of Chapter 2, one might wonder in what way the new tiered augmenting paths are better than the shortest augmenting paths, in what way the relaxed tier $\mathrm{tier}^\star_t(w)$ improves the plain distance from the seeds, i.e., $\mathrm{dist}_{G_n \bowtie M_t}(w, \mathcal{S}_t)$. In fact, it is exactly Properties 3 and 4 that differentiate the two—when orientation of edges changes, the length of the shortest path from w to $\mathcal{S}_t$ may increase even by a factor of two, while the growth of ranks, and thus also tiers, is bounded both in terms of neighbors (Property 3) and the last augmenting path (Property 4). In result, the relaxed ranks and tiers are bounded by $\mathcal{O}\left(|V|^{1/2}\right)$, while the distance to $\mathcal{S}_t$ might be easily linear in $|V|$.

Despite the relaxed rank and tier being more flexible than the original concepts from Definitions 3.1 and 3.2, all the following claims still hold. In all of them we assume that $\mathrm{rank}^\star$ and $\mathrm{tier}^\star$ are relaxed rank and tier in the sense of Definition 3.28. In most cases the proofs need only slight changes, therefore, we omit arguments which require just straightforward modifications. Nevertheless, the reasoning for Lemma 3.30 is more complicated, and included in full. Furthermore, it may not be entirely clear how to transform the last paragraph of the proof of Lemma 3.14, so for Lemma 3.34 we include a detailed sketch of that part.

Lemma 3.29. *Suppose that the sequence of matchings $M_1, M_2, \ldots$ was constructed by applying each turn an augmenting path which was tiered. Then, for any vertex $v \in V(G_n)$ its tier $\mathrm{tier}^\star_t(v)$ is a non-decreasing function of t.* □

Lemma 3.30. *For any edge $e \in G_n{}^{\bowtie M_t}$ we have*

$$\begin{aligned} \operatorname{rank}^\star_t(\operatorname{tail}(e)) &\leqslant \operatorname{tier}^\star_t(\operatorname{head}(e)) + 1 && \text{if } e \text{ is matched in } t, \\ \operatorname{tier}^\star_t(\operatorname{tail}(e)) &\leqslant \operatorname{tier}^\star_t(\operatorname{head}(e)) && \text{otherwise.} \end{aligned}$$

Proof. The second inequality, as in Lemma 3.10, follows trivially from the definition of $\operatorname{tier}^\star$.

To prove the first inequality, we proceed inductively on t. Since e is matched in t, we set $w = \operatorname{tail}(e)$. Surely, the bound is satisfied in $t = 0$, because both the rank and the tier are 0. Now, if $w \notin V(A_t)$, then e is matched in $t-1$ and $b^t_w = b^{t-1}_w$, thus

$$\begin{aligned} \operatorname{rank}^\star_t(w) &\overset{(3)}{\leqslant} \max\{\operatorname{tier}^\star_{t-1}(b^t_w) + 1,\ \operatorname{tier}^\star_{t-1}(w)\} \\ &\overset{\text{def.}}{\leqslant} \max\{\operatorname{tier}^\star_{t-1}(b^t_w) + 1,\ \operatorname{rank}^\star_{t-1}(w),\ \operatorname{tier}^\star_{t-1}(b^{t-1}_w)\} \\ &\overset{\text{ind.}}{\leqslant} \max\{\operatorname{tier}^\star_{t-1}(b^t_w) + 1,\ \operatorname{tier}^\star_{t-1}(b^{t-1}_w) + 1,\ \operatorname{tier}^\star_{t-1}(b^{t-1}_w)\} \\ &\leqslant \operatorname{tier}_t(b^t_w) + 1. \end{aligned}$$

On the other hand, if $w \in V(A_t)$, then because the augmenting path A_t is tiered we have $\operatorname{tier}^\star_{t-1}(b^t_w) \geqslant \operatorname{tier}^\star_{t-1}(w)$, hence

$$\operatorname{rank}^\star_t(w) \overset{(3)}{\leqslant} \operatorname{tier}^\star_{t-1}(b^t_w) + 1 \leqslant \operatorname{tier}^\star_t(b^t_w) + 1. \qquad \square$$

Corollary 3.31. Let P be a seeded path, then every white vertex $w \in W(P)$ of rank $\operatorname{rank}^\star_t(w) \geqslant 1$ is followed by some vertex $w' \in W(P)$ of rank $\operatorname{rank}^\star_t(w') \geqslant \operatorname{rank}^\star_t(w) - 1$. ▫

Corollary 3.32. The length of the shortest path from $v \in V(G_n)$ to a seed in $G_n{}^{\bowtie M_{t-1}}$ is at least $2\operatorname{tier}^\star_{t-1}(b)$. ▫

Corollary 3.33. For any t and a non-empty set $\mathcal{P}$ of vertex-disjoint seeded paths of finite rank in $G_n{}^{\bowtie M_t}$

$$\min\{\operatorname{rank}^\star_t(P) \mid P \in \mathcal{P}\} \leqslant \frac{|M_t \cap E(\mathcal{P})|}{|\mathcal{P}|}. \qquad \square$$

Lemma 3.34. *Let $Q_1, Q_2, \ldots, Q_l$ be a sequence of vertex-disjoint seeded paths in ${G_t}^{\bowtie M_t}$. Also, let $b_t \xrightarrow{A_t} s$ be the tiered augmenting path in turn t, i.e., in ${G_t}^{\bowtie M^{t-1}}$. Then, there exist vertex-disjoint seeded paths $R_1, \ldots, R_l$ and P in ${G_{t-1}}^{\bowtie M_{t-1}}$ such that*

$$\begin{aligned} &\operatorname{rank}^\star_{t-1}(A_t) \leqslant \operatorname{rank}^\star_{t-1}(P), \\ &\quad \operatorname{rank}^\star_t(Q_i) \leqslant \operatorname{rank}^\star_{t-1}(R_i) && \text{for } \operatorname{rank}^\star_t(Q_i) > \operatorname{rank}^\star_{t-1}(A_t) + 1, \\ &\quad \operatorname{rank}^\star_t(Q_i) \leqslant \operatorname{rank}^\star_{t-1}(R_i) + 1 && \text{otherwise.} \end{aligned}$$

As it is only the last paragraph of the original proof that changes significantly, we sketch below only that particular part.

Sketch of the last paragraph. Consider a path Q_i from Lemma 3.14 and let w_i be the highest-ranked vertex of Q_i, namely $\operatorname{rank}_t(Q_i) = \operatorname{rank}_t(w_i)$. Now suppose that $\operatorname{rank}_t(w_i) > \operatorname{rank}_{t-1}(A_t) + 1$, that is, we need the corresponding path R_i, which connects w_i to some seed at turn $(t-1)$, to satisfy $\operatorname{rank}_t(Q_i) \leqslant \operatorname{rank}_{t-1}(R_i)$. However,

$$\operatorname{rank}_t(Q_i) = \operatorname{rank}_t(w_i) \overset{(4)}{\leqslant} \operatorname{tier}_{t-1}(w_i) \leqslant \operatorname{rank}_{t-1}(R_i).$$

Otherwise, for $\operatorname{rank}_t(w_i) \leqslant \operatorname{rank}_{t-1}(A_t) + 1$ we have

$$\begin{aligned} \operatorname{rank}_t(Q_i) = \operatorname{rank}_t(w_i) &\overset{(3)}{\leqslant} \max\left\{\operatorname{tier}_{t-1}(b^t_{w_i}) + 1,\ \operatorname{tier}_{t-1}(w_i)\right\} \\ &\overset{(\spadesuit)}{\leqslant} \operatorname{tier}_{t-1}(w_i) + 1 \\ &\leqslant \operatorname{rank}_{t-1}(R_i) + 1. \end{aligned}$$

If $w_i \notin V(A_t)$, then the inequality marked by $(\spadesuit)$ follows from $b^t_{w_i} = b^{t-1}_{w_i}$. On the other hand, because the augmenting path A_t is tiered, $w_i \in V(A_t)$ implies that w_i is the vertex of the smallest tier among out-neighbors of $b^t_{w_i}$, so $\operatorname{tier}_{t-1}(b^t_{w_i}) = \operatorname{tier}_{t-1}(w_i)$. □

It might seem that the above reasoning could be simplified if the first argument of max in Property 3 was $\mathrm{tier}_{t-1}(w)+1$ or $\mathrm{tier}_{t-1}(b_w^{t-1})+1$, yet, neither does work. The former fails, because $\mathrm{tier}_{t-1}(w) \geqslant \mathrm{rank}_{t-1}(w)$, that is, given high enough ranks of augmenting paths, $\mathrm{rank}(w)$ could raise beyond the limit implied by Lemma 3.10. Furthermore, the latter is not valid in a case where $w \in V(A_t)$ and $\mathrm{tier}_{t-1}(w) > \mathrm{tier}_{t-1}(b_w^{t-1})$—the increase by 1 is not enough to accommodate the growth of rank due to the augmenting path.

Lemma 3.35. *For any turn $\tau \in \{1,2,\ldots,n\}$, any white vertex $w \in W_\tau$ and a natural number $j \in \{1,2,\ldots,\mathrm{rank}^\star_\tau(w)\}$, there is a turn $t < \tau$ and a collection $\mathcal{B}_t$ of j pairwise vertex-disjoint seeded simple paths $\mathcal{B}_t = \{\mathcal{K}_1,\ldots,\mathcal{K}_j\}$ in $G_t{}^{\bowtie M_t}$ such that*

$$\forall \mathcal{K} \in \mathcal{B}_t.\ \mathrm{rank}^\star_t(\mathcal{K}) \geqslant \mathrm{rank}^\star_\tau(w) - |\mathcal{B}_t|. \qquad \square$$

Theorem 3.36. *For any dynamic unweighted matching algorithm that uses tiered augmenting paths it holds that $\mathrm{rank}^\star_n(w) \leqslant 2|M_n|^{1/2}$ for every $w \in W$.* $\square$

Corollary 3.37. For any $v \in V(G_n)$ we have $\mathrm{tier}^\star_n(v) \leqslant 2|M_n|^{1/2}$. $\square$

In other words, any sequence of functions that each turn assign natural numbers to white vertices, not necessarily closely related to augmenting paths, could be relaxed ranks—as long as they satisfy the properties listed in Definition 3.28, their values are bounded from above by $\mathcal{O}\big(|M_n|^{1/2}\big)$. Thanks to that, in Chapter 4 we will be able to establish that the running time of the searching procedure Algorithm 1 is $\mathcal{O}\big(|E(G_n)| \cdot |M_n|^{1/2}\big)$.

Nonetheless, even for relaxed ranks the corresponding tiers may still require costly updates. In order to address this problem we will use tier lower bounds rather than their exact values, i.e., some non-decreasing natural numbers smaller or equal to the actual tier. The intuition behind this is that we do not really need to calculate how high the exact values of tiers are—when guiding the algorithm it is enough to know if the tier is lower, or higher than the current path-searching threshold, that is, whether we do or do not want the search to explore the vertex in question. This way it is possible to update the bounds lazily, only when the current value

of some vertex's tier lower bound is smaller than or equal to the current threshold.

Still, for Theorem 3.36 to work, it is crucial for the tiered augmenting paths to follow the true relaxed tiers of vertices, and not just their lower bounds. To make sure this is the case, we would like to precede any search for an augmenting path with a search for a witness of the value of the current tier lower bound. If there is no such witness, then it is a proof that the tier lower bound is too low, and so we should increase its value. On the other hand, if the witness is found, it is a proof that the lower bound matches the actual tier and it is safe to look for an augmenting path. However, observe that the augmenting path we are searching for is also precisely the witness for $\mathrm{tier}_{t-1}(b_t)$ we would like to have. If such path is found, all the necessary invariants have to be satisfied, which ensures that Theorem 3.36 will work.

The details of the searching strategy sketched above are the subject of Section 4.1—this procedure forms the basis of all the matching algorithms of Chapter 4. We end this part with an observation which greatly simplifies the design as well as the analysis of the aforementioned algorithms. More precisely, as a direct consequence of how $\mathrm{tier}^\star_t$ is defined, there is a very convenient candidate for the tier lower bound.

Observation 3.38. The following two inequalities give a lower bound on the $\mathrm{tier}^\star_t$:

$$\begin{aligned} &\mathrm{tier}^\star_t(w) \geqslant \mathrm{rank}^\star_t(w) && \text{for any } w \in W(G_n), \\ &\mathrm{tier}^\star_t(b) \geqslant \min\Big\{\mathrm{rank}^\star_t(w) \;\Big|\; \langle b, w\rangle \in E\big(G_n{}^{\bowtie M_t}\big)\Big\} && \text{for any } b \in B(G_n). \end{aligned}$$ □

Chapter 4

Bipartite Matching Algorithms

In this chapter we present the algorithms that build upon the technique of ranks and tiers introduced in Chapter 3. In the relaxed form presented in Section 3.5, they apply to a variety of approaches—we consider online and offline settings, both either exact or approximate. At the core of all the algorithms presented here is the search procedure described informally at the end of Section 3.5. The details of this fundamental part are given in the first section of this chapter.

The main result is the algorithm for the online case, described in Section 4.2, together with the corresponding approximation algorithm. After that, we consider the easier offline case. We compare our approach with two bipartite matching methods: the well-known Hopcroft-Karp algorithm [59], and the auction-based algorithm of Demange, Gale, and Sotomayor [28]. We end this chapter exploring some additional settings, i.e., weighted graphs and vertex-decremental dynamic matching.

To reflect the algorithmic nature of this chapter, we drop the turn indexes from G_t and M_t, that is, we refer by G and M to the current graph and matching respectively. Moreover, we handle the ranks and tiers similarly, using $w.\mathrm{rank}^*$ for any white vertex $w \in W(G)$ and writing $w.\mathrm{rank}^* \leftarrow w.\mathrm{rank}^* + 1$ to indicate the updates.

Still, to apply the theorems from the previous chapter we occasionally need to ground ourselves in turns or use turn-indexed ranks and tiers. To this end, we define $\mathrm{rank}^\star_t(w)$ as the $w.\mathrm{rank}^*$ immediately after

the t'th augmenting path A_t has been applied, and denote by $\text{tier}^\star$ the corresponding tiers.

Definition 4.1. For any white vertex w and its counter $w.\text{rank}^*$ maintained by an algorithm we define

$$\text{rank}^\star_t(w) \stackrel{\text{def}}{=} w.\text{rank}^*$$

where $w.\text{rank}^*$ is taken at the time of computation precisely after the t'th augmenting path has been applied. We set $\text{tier}^\star_t$ to be the tiers defined similarly to Definition 3.2 only using $\text{rank}^\star_t$ as the underlying ranks. ◇

At the end of the next section we show in Lemma 4.2 that ranks defined in this way by our algorithms indeed satisfy the conditions of the relaxed ranks from Definition 3.28. This allows us to employ the results derived in Chapter 3 to bound, using the values of $\text{rank}^\star$ and $\text{tier}^\star$ defined above, the total length of augmenting paths and the running time of algorithms that produced them.

Results included in this chapter appeared first at FOCS 2014 [18].

4.1 Searching

The SEARCH procedure constitutes the base component of the matching algorithms, which rely on the ranks and tiers. Although it can be implemented in multiple ways—we could use the framework of practically any graph-searching method like the depth-first search, breadth-first search, heuristic searches, or schemas based on randomized priorities, etc.—these would all be just different flavors of the same algorithm. Hence, for the sake of simplicity of the exposition we use the DFS-based version.

The purpose of calling SEARCH(w) is twofold:

- if the lower bound matches the actual tier, i.e., $w.\text{rank}^* = \text{tier}^\star_{t-1}(w)$, it finds a tiered path certifying that this is the case;
- otherwise, if $w.\text{rank}^* < \text{tier}^\star_{t-1}(w)$, it increases the ranks (that is, the tier lower bounds) that are too low.

The idea is to follow, as long as it is possible, the edges that appear tiered according to the lower bounds on the $\text{tier}^\star$'s, for which we use the rank^*'s, as pointed out in Observation 3.38. Should it happen that we reach a seed, then the path we have used constitutes a witness for the rank^*'s to be equal to the $\text{tier}^\star$'s, which is what our procedure was searching for. On the other hand, if the search tree leads to a vertex from where no tiered path can be found, it means that the lower bounds of all the visited vertices were strictly below the $\text{tier}^\star$'s. Thus, we increase the relevant counters, that is, the rank^*'s. The pseudocode is presented below.

Algorithm 1 The searching procedure

```
1:  procedure SEARCH(w)
2:      w.rank* ← w.rank* + 1
3:      for each b such that ⟨b, w⟩ ∈ G^⋈M do
4:          UPDATEINFORMATION(b, w)
5:      if b_w = ⊥ then
6:          return true
7:      w' ← SMALLESTNEIGHBORUNMATCHEDTO(b_w)
8:      while w'.rank* < w.rank* do
9:          if SEARCH(w') then
10:             M ← M \ {⟨w, b_w⟩} ∪ {⟨b_w, w'⟩}
11:             UPDATEINFORMATION(b_w', w)
12:             UPDATEINFORMATION(b_w', w')
13:             return true
14:         w' ← SMALLESTNEIGHBORUNMATCHEDTO(b_w)
15:     return false
```

It is easy to observe that the listing does not agree perfectly with the description. For example, in line 2 we increase the ranks whether the search was successful or not, while in line 10 we are applying the augmenting path. Indeed, the pseudocode implements the algorithm using some tricks to make the code simpler and shorter. In particular, in line 10 we take advantage of the recursion—because all the given algorithms are matching procedures, we always need to apply the augmenting path after each successful search. Moreover, as mentioned, in line 2 we increase the rank of a vertex even before a witness is found or we know that there is

no such path. This is alright, because in both cases the rank finally gets increased, we just do it ahead of time. The benefit of such solution is that we do not have to maintain explicitly which vertices were already visited—the algorithm is only allowed to explore vertices of smaller ranks, so due to the rank increase in line 2 and condition in line 8, it cannot revisit a vertex that already belongs to the path currently being explored.

We now move on to describe the subroutines used in the pseudocode and their complexity. Observe that since Search is also only a subprocedure of algorithms presented later in this chapter, we actually do not know how many times it and its components will be called. However, thanks to line 2, the rank^*'s provide the bound we need—we can express the running times with respect to $\mathcal{R}$, the maximum rank achieved by the hypothetical, yet undefined algorithm. In this way our analysis is applicable to any algorithm which relies on Search to find augmenting paths and manage the rank^*'s.

The call SmallestNeighborUnmatchedTo(b_w), which happens before each run of the loop, i.e., in lines 7 and 14, returns an out-neighbor of b_w with the smallest current rank^*. This subprocedure can be implemented to work in constant time by maintaining for each black vertex $b \in B(G)$ a list of out-neighbors of the smallest rank^*,

$$b.\text{list} = \left\{ w \in \mathcal{N}_{G^{\bowtie M}}(b \to) \;\middle|\; \forall w' \in \mathcal{N}_{G^{\bowtie M}}(b \to).\ w.\text{rank}^* \leqslant w'.\text{rank}^* \right\}.$$

Then, SmallestNeighborUnmatchedTo(b_w) simply returns any white vertex that is an element of $(b_w).\text{list}$.

To keep these lists up to date we use another subroutine, namely UpdateInformation(b, w). Its task is to maintain the presence of w on $b.\text{list}$. Each time the rank of a white vertex w increases, it needs to be removed from the lists of all its in-neighbors (see line 4). Clearly this happens at most $\mathcal{R}$ times per edge, and requires $\mathcal{O}(|E| \cdot \mathcal{R})$ time over the whole algorithm run. Moreover, at any given time, if a black vertex b finds its list of the smallest out-neighbors empty, $b.\text{list} = \varnothing$, it scans all the neighbors and repopulates the list with white vertices of the smallest rank^*. The empty list indicates that the rank^*'s of all neighbors of b increased, and so

its tier lower bound had to increase as well (see Observation 3.38). Therefore, for any black vertex b its list repopulation happens at most $\mathcal{R}$ times, taking $\mathcal{O}(\mathcal{R} \cdot \deg(b))$ time per vertex, or $\mathcal{O}(|E| \cdot \mathcal{R})$ in total. There are also additional updates necessary each time an edge changes direction. Their cost however is covered by the sum of lengths of all the augmenting paths, which is bounded by $\mathcal{O}(|V| \cdot \mathcal{R})$.

All the described maintenance operations are performed in Algorithm 1 by UpdateInformation calls in lines 4, 11 and 12. Combining the involved complexity bounds yields $\mathcal{O}(|E| \cdot \mathcal{R})$ time for the work done while updating. This is also the total running time of all the calls to Search procedure—the loop in line 3 does $\deg(w)$ iterations, while any other yet unaccounted line takes constant time—all the parts likewise sum up to $\mathcal{O}(|E| \cdot \mathcal{R})$.

As stated at the beginning of this section, Search constitutes the base of all the matching algorithms in this chapter. In fact, its core, that is, the code between lines 7 and 14 of Algorithm 1, will later become useful by itself. To this end, we extract it into a separate auxiliary procedure. The purpose of MatchUntil(b, r) is to try to find an augmenting path from b until all its neighbors reach the given threshold r. Because we always start searching at some unmatched black vertex, and apply the augmenting path if the search was successful, all first calls to Search are performed by MatchUntil.

Algorithm 2 The MatchUntil auxiliary procedure

```
1: procedure MatchUntil(b, r)
2:     w' ← SmallestNeighborUnmatchedTo(b)
3:     while w'.rank* < r do
4:         if Search(w') then
5:             M ← M ∪ {⟨b, w'⟩}
6:             UpdateInformation(b, w')
7:             return true
8:         w' ← SmallestNeighborUnmatchedTo(b)
9:     return false
```

To finalize the exposition of Algorithm 1, we still need to explain why the $\text{rank}^\star_t$'s produced via rank^*'s by an algorithm which uses Search to

find augmenting paths and manage its ranks, satisfy the four properties of Definition 3.28. However, since the ranks are maintained by SEARCH, it is enough to show that consecutive applications of the procedure generate ranks that are appropriate.

For convenience, we restate the conditions set on relaxed ranks below:

1. $\mathrm{rank}^\star_t(w)$ is non-decreasing in t,
2. $\mathrm{rank}^\star_t(w)$ increases when an augmenting path uses w,
3. $\mathrm{rank}^\star_t(w) \leqslant \max\{\mathrm{tier}^\star_{t-1}(b^t_w) + 1,\ \mathrm{tier}^\star_{t-1}(w)\}$,
4. $\mathrm{rank}^\star_t(w) \leqslant \max\{\mathrm{rank}^\star_{t-1}(A_t) + 1,\ \mathrm{tier}^\star_{t-1}(w)\}$.

Lemma 4.2. *If the ranks are managed solely by* SEARCH *and its components, then the* $\mathrm{rank}^\star$ *from Definition 4.1 and its corresponding* $\mathrm{tier}^\star$ *are* relaxed *in the sense of Definition 3.28, that is, they satisfy the four properties reproduced above.*

Proof. As all the changes to the ranks are done in line 2 of Algorithm 1, Properties 1 and 2 are clearly true. To arrive at the last two points, we separately consider the rank growth due to an unsuccessful search, and the rank increase caused by a search that ended with a success.

Let A_{t-1} be the last augmenting path applied before the search starts. Observe that for each vertex w whose rank increased when the outer-most call to search failed, it holds that $w.\mathrm{rank}^* < \mathrm{tier}^\star_{t-1}(w)$, and so $w.\mathrm{rank}^* \leqslant \mathrm{tier}^\star_{t-1}(w)$ after the update. Note also that rank increase resulting from a failed search does not affect any $\mathrm{tier}^\star$'s.

In turn, suppose that $w.\mathrm{rank}^*$ was increased during the search that found A_t. Although we assume the outer-most search was successful, the procedure could perform a number of inner recursive subsearches, some of which may have failed. Similarly, as in the previous paragraph, that could happen because the actual tier was strictly higher than the rank^* of the explored vertex. However, there is a second possibility—the tier lower bounds that blocked our way could have been increased during the current search. For that reason, the argument in the successful case has to be a bit more complicated.

Observe that the rank update together with $\mathrm{tier}^\star_{t-1}(b_w^{t-1}) < \mathrm{rank}^\star_{t-1}(w)$ implies $w \in V(A_t)$ and $\mathrm{tier}^\star_{t-1}(b_w^t) = \mathrm{rank}^\star_{t-1}(w)$. That implies $w.\mathrm{rank}^* \leqslant \mathrm{tier}^\star_{t-1}(b_w^t) + 1$. On the other hand, if $\mathrm{tier}^\star_{t-1}(b_w^{t-1}) \geqslant \mathrm{rank}^\star_{t-1}(w)$, then

$$\begin{aligned} w.\mathrm{rank}^* &\leqslant \mathrm{tier}^\star_{t-1}(b_w^{t-1}) + 1 = \mathrm{tier}^\star_{t-1}(b_w^t) + 1 && \text{if } b_w^t = b_w^{t-1}, \\ w.\mathrm{rank}^* &\leqslant \mathrm{tier}^\star_{t-1}(b_w^{t-1}) + 1 \leqslant \mathrm{tier}^\star_{t-1}(b_w^t) + 1 && \text{if } w \in V(A_t). \end{aligned}$$

Furthermore, because the search was successful, due to the check at line 8 we have that $w.\mathrm{rank}^* \leqslant \mathrm{tier}^\star_{t-1}(b_t) + 1 = \mathrm{rank}^\star_{t-1}(A_t) + 1$.

Finally, if some rank does not change from turn $t-1$ to t, then trivially $w.\mathrm{rank}^* = \mathrm{rank}^\star_{t-1}(w) \leqslant \mathrm{tier}^\star_{t-1}(w)$. Together, the bounds on $w.\mathrm{rank}^*$ give the desired bounds on $\mathrm{rank}^\star_t(w)$. □

4.2 Online Matching

The apparatus of the ranks and tiers developed in this thesis culminates in Algorithm 3 shown in the listing below. It solves the problem of maintaining a maximum cardinality matching in an incremental setting defined in Definition 1.9.

In short, we are given a bipartite graph $G = \langle W \uplus B, E \rangle$ in a one-sided online fashion, that is, all the vertices of W are known from the start, while vertices of B are given one by one during the algorithm run—each turn a new vertex $b_t \in B$ arrives with all its incident edges. At that time the call to $\textsc{Match}(b_t)$ produces the matching $\mathcal{M}$, which is the maximum cardinality matching in the part of G presented up to this point.

Due to the analysis in Chapter 3 its total running time is $\mathcal{O}\big(|E| \cdot |V|^{1/2}\big)$. The procedure MatchUtil allows us to formulate it very concisely, the two lines constitute the whole listing.

Algorithm 3 The exact algorithm for the problem from Definition 1.9

```
1: procedure Match(b_t)
2:     MatchUntil(b_t, 2 · |M|^{1/2})
```

Although the algorithm is quite simple, to the best of author's knowledge at the time of writing there are no other procedures for this particular

setting with running time better than $\mathcal{O}(|E| \cdot |V|)$, which is the bound of the naïeve augmenting-path approach. Algorithm 3 leads to the following theorem.

Theorem 4.3. *There exists an algorithm that maintains an exact maximum cardinality matching for the problem setting in Definition 1.9 in $\mathcal{O}(|E| \cdot |V|^{1/2})$ total time.* □

The same bounds that imply the running time allow us to bound also the total length of all the augmenting paths applied. Although it is worse than the logarithmic conjecture of Chaudhuri, Daskalakis, Kleinberg and Lin in [24], it is the only bound for the general bipartite case better than the basic $\mathcal{O}(|V|^2)$.

Observation 4.4. The order of the total length of all the augmenting paths applied by Algorithm 3 is $\mathcal{O}(|V|^{3/2})$. □

Next we present a corresponding approximation algorithm, which for any constant $\varepsilon > 0$ maintains a $(1-\varepsilon)$-approximate matching and works in $\mathcal{O}(|E| \cdot \varepsilon^{-1})$ time. This algorithm does not use the bound on the maximum rank of Theorems 3.16 and 3.36, instead it leverages the fact that ranks bound the lengths of augmenting paths from below. More precisely, the invariant is that the tier of a black vertex that does not get matched is at least ε^{-1}. This implies, due to Observation 1.4, that the algorithm achieves $(1-\varepsilon)$-approximation of the optimal matching at any given turn.

Algorithm 4 The $(1-\varepsilon)$-approximation algorithm for the same problem

```
1: procedure MatchApprox(ε, b_t)
2:     MatchUntil(b_t, ε^{-1})
```

Similarly to the exact procedure, the approximation algorithm also has two claims analogous to Theorem 4.3 and Observation 4.6.

Theorem 4.5. *There exists an algorithm that maintains a $(1-\varepsilon)$-approximate maximum cardinality matching for the problem setting in Definition 1.9 in $\mathcal{O}(|E| \cdot \varepsilon^{-1})$ total time.* □

Observation 4.6. The order of the total length of augmenting paths applied by Algorithm 4 is $\mathcal{O}(|V| \cdot \varepsilon^{-1})$. □

To better explain and provide additional intuition on how Algorithm 3 works, we look into three sample calls of MATCH on an instance of a graph pictured in Figure 4.1a, namely MATCH(b_A), MATCH(b_B) and MATCH(b_E).

The numbers in the white vertices are their current rank*'s, while the values of $\text{tier}^\star_{t-1}$'s, that is, the relaxed tiers from when the previous turn has ended, are indicated by the background shades enclosed by the dotted line. As the maximum rank* is only 1, there are only three possible values of relaxed tiers, namely 0, 1 and ∞, denoted respectively by absence of a shade, a light shade and a darker shade. For convenience, these numbers are repeated near each black vertex, for example $\text{tier}^\star_{t-1}(b_C) = \infty$ despite the fact the minimum over the rank*'s of neighbors is only 1.

The final graph has 14 black and 14 white vertices. In Figure 4.1a we are in turn 10, that is, 9 black vertices have been already matched. There are still five seeds left, labeled respectively w_A, w_B, w_C, w_D, w_E and five more black vertices b_A, b_B, b_C, b_D, b_E to add. To make them easily distinguishable their edges were drawn as curvy, dashed lines.

The vertex we are going to match in turn 10 is b_A, marked with a small additional circle around it. The pink area surrounded by the dot-dashed line is the graph reachable from b_A by the edges that appear tiered according to rank*'s. For example, the edge marked by letter e is tiered neither according to rank*'s, nor to the $\text{tier}^\star_9$'s. To make the example interesting, we match b_A to w_A.

The second diagrams shows the next turn $t = 11$, in which we are going to try to match vertex b_B. Of the three available tiered augmenting paths we pick the shorter of two that lead to w_B. Despite the path to w_C is shorter, the algorithm does not know it, nor does it even take it into account—it starts exploration in the direction of the neighbor of the smallest rank*, and the one on its left satisfies all the necessary conditions.

There is also a path that leads to the dead area. If the algorithm were to choose it first, the only effect would be that the rank*'s along that path would have been increased. We will explore this possibility when analyzing the search calls for MATCH(b_E), now we would like only to note

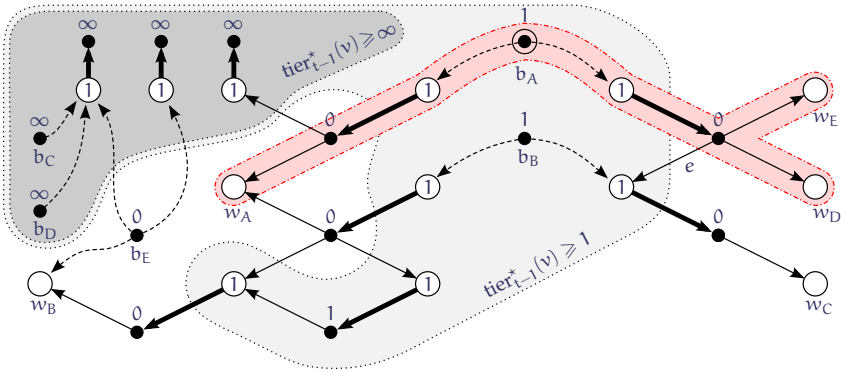

Figure 4.1a: The tiered augmenting paths starting at b_A.

that the edge that leads from w_A to $b^{10}_{w_A}$ could have been used by the search this turn. As the rank* of the white vertex in the dead area is too low, the edge appears tiered according to rank*'s, even if it is not tiered with regard to $\mathrm{tier}^\star_{10}$'s.

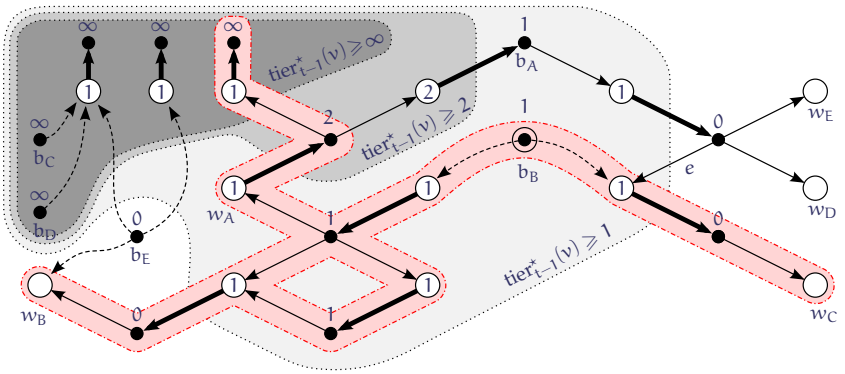

Figure 4.1b: Vertices reachable by the tiered search starting at b_B.

To see how Search works in more details, consider the Figure 4.1c. Every time we enter a new vertex, line 2 in Algorithm 1 increases the rank*'s of visited vertices. For example, suppose that in turns 12 and 13 we have added b_C and b_D which both failed to match. The search increased the rank* of the encountered white vertex from 1 to 2 and to 3. Of course, even after that, the value of rank* of that white vertex still bounds its relaxed tier from below.

Consider now the final turn $t = 14$ in which we add b_E. When we start the search from b_E, because of the difference in rank*'s, the edge to the

white vertex from previous paragraph is not available, as indicated by the short red mark on it. Fortunately, b_E has two more neighbors, both of small rank*. As we enter w_B, due to line 2 in Algorithm 1 we increase its rank* and proceed ahead. When the search fails after encountering a vertex of bigger rank* (second red mark), we know that the previous w_B. rank* was too small—we consider the traversed path as a way of updating the ranks of the vertices along it.

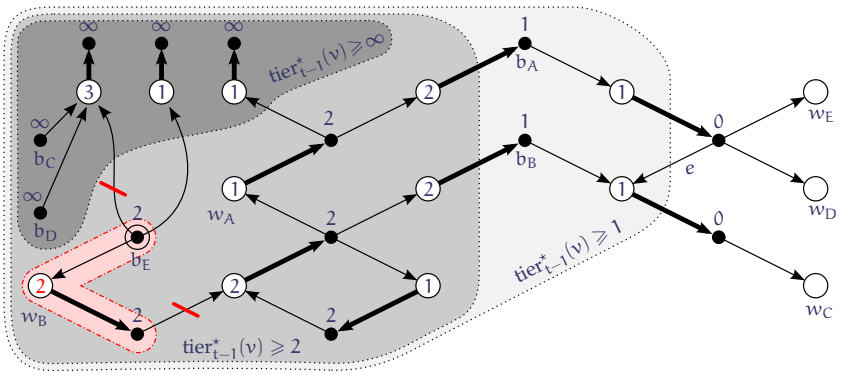

Figure 4.1c: A single step of Algorithm 1 starting at b_E. Even if the search finds the correct paths, it fails because w_B. rank$^* < \text{tier}^\star_{t-1}(w_B)$.

After we return from the first call to Search at line 4 in Algorithm 2, we find ourselves back at b_E. There is another neighbor of tier$^* = 1$, but the search also fails. Note, that despite our previous visit to w_B, we do not consider it visited—merely updated— because of the update we performed previously, its tier* is now 2. The current search reached a dead region and ends with the same conclusion.

In Figure 4.1e we start searching again in the direction of w_B, but as w_B. rank$^* = \text{tier}^\star_{t-1}(w_B)$, this call to Search should find a witness. We increase all the rank*'s of the vertices visited along the way. Naturally, we may hit a dead end, and need to retract some of our steps. Recall that the search follows the vertices of the smallest tier*'s, which might make some edges unavailable, despite them being tiered according to $\text{tier}^\star_{t-1}$. Nevertheless, it is not necessary to back out completely, just enough to find another neighbor of low rank*.

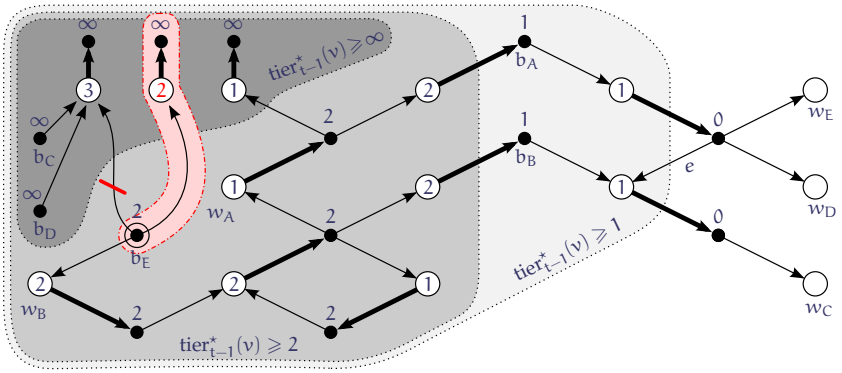

Figure 4.1d: Because of outdated rank^*'s, the search might explore a direction that does not lead to any seed.

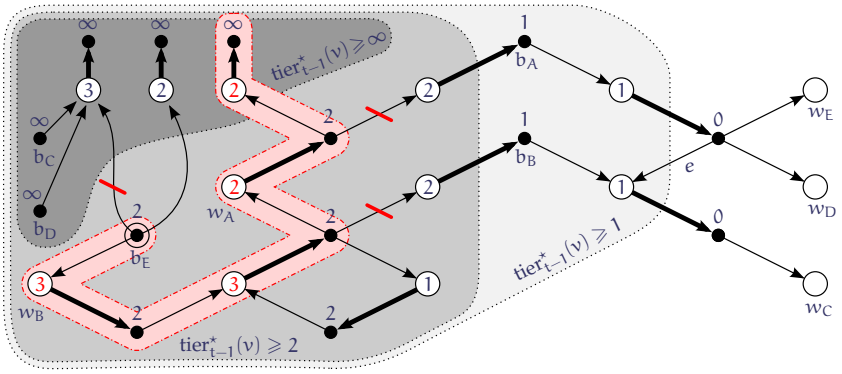

Figure 4.1e: Although w_B has already been visited, we do it again, each time increasing $w_B.\text{rank}^*$.

After retracting our steps we pick the other neighbor of $\text{rank}^* = 1$, and, unfortunately, hit another dead end. In fact, it would have been a dead end even if we were at the rank level 2—the rank^*'s of already visited vertices increased to 3 prevent us from creating a non-simple path.

Nonetheless, it is worth noting how we consider w_A as not visited, just updated, similarly to what happened in Figure 4.1d. Yet, because we did not backtrack all the way to the start, its predecessors on the search path are still marked by the red hue. This is because these vertices are present on the stack of recursive calls of SEARCH, while w_A is not. Also, due to its updated rank^*, the path to w_A is now blocked.

In Figure 4.1g we finally manage to find the augmenting path. Despite the predecessor of w_A having a neighbor of the smallest rank^* which have

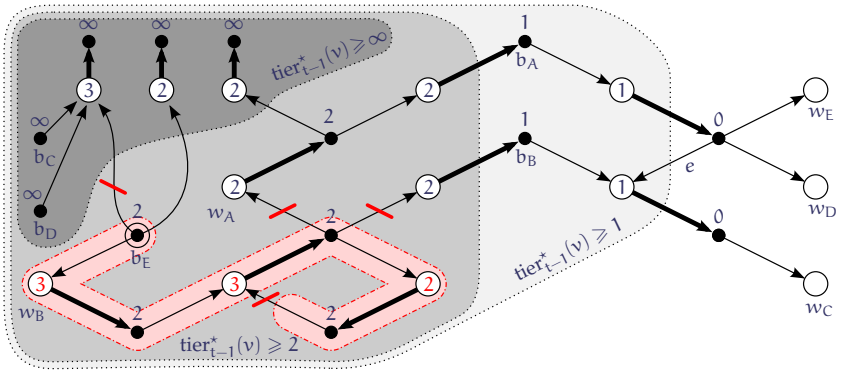

Figure 4.1f: As only the inner recursive calls of Algorithm 1 have failed, rather than starting from scratch we continue the previous search.

never been previously considered in any search calls of $\textsc{Match}(b_E)$, the vertex w_A also had the smallest rank*, and so could have been picked.

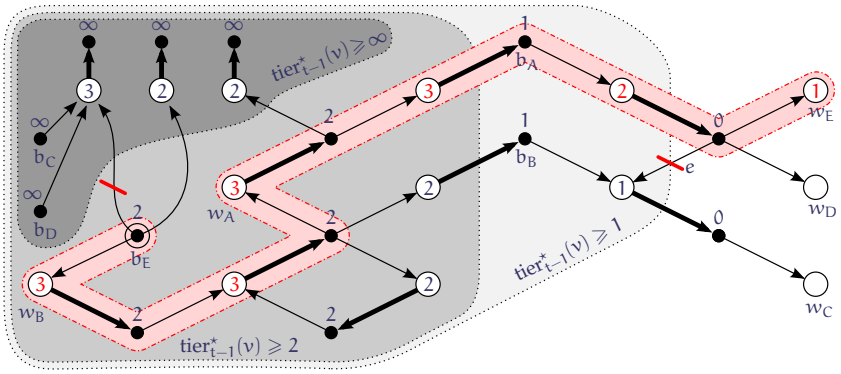

Figure 4.1g: Finally, the tiered augmenting path has been found.

The last diagram presents the situation after all the updates and reversing of edges. Half of the graph is dead, and the other half has tiers ranging from 0 to 3. Not all the rank*'s are up to date, even vertices of the augmenting path may have rank$^* < \text{tier}^\star_{14}$. We have failed to match b_C and b_D, and so two white vertices w_C and w_D are still seeds after the process.

As Figure 4.1g demonstrates, there is some leeway in picking the neighbors of the smallest rank form b.list. Nonetheless, the form of the recursive calls makes the DFS structure in Algorithm 1, i.e., that particular implementation of Search procedure, quite apparent. For example, the combi-

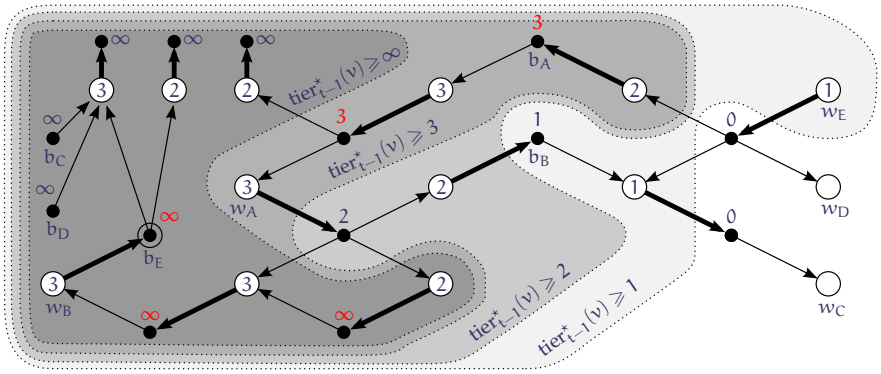

Figure 4.1h: After the augmenting path has been applied, the tiers increase.

nation of the rank increase in line 2 and loop condition in line 8, is the direct equivalent of marking the vertices visited in the DFS. As mentioned above, it is impossible for any vertex to be used more than once in a single path, and so the witness, if one is found, has to be simple. On the other hand, the behavior of Algorithm 1 is quite constrained. Especially, because the dynamic setting forces the order of vertices we attempt to match.

For this reason, in order to better compare our strategy to other existing matching algorithms, in the next section we are going to consider the case when the whole graph is given up front. In the offline setting we can choose which black vertex we would like to match—even sticking to the DFS-based flavor, a matching procedure based on the techniques from Chapter 3 can be tweaked in many different ways. For example, it can be made to mimic some aspects of the Hopcroft-Karp algorithm. This intrinsic flexibility of the ranks and tiers is a big advantage of that approach.

4.3 Offline Matching

One of the best-known matching algorithms is the Hopcroft-Karp algorithm which applies to unweighted bipartite graphs in an offline setting [59]. Although it relies on the shortest augmenting paths, just as the algorithms in Chapter 2, its $\mathcal{O}(|E| \cdot |V|^{1/2})$ complexity stems from application of a number of disjoint paths per round. In the k-th round the

Hopcroft-Karp algorithm finds a maximal set of disjoint augmenting paths of length $(2k-1)$. It is shown that if the collection of the disjoint paths cannot be extended by any other path of that length, the length of the shortest augmenting path in the graph strictly increases. This is why there are at most $\mathcal{O}\left(|V|^{1/2}\right)$ rounds, each of which takes $\mathcal{O}\left(|E|\right)$ time.

A similar thing can be achieved using ranks and tiers. We set $r = 0$ and start with a FIFO queue which contains all the unmatched black vertices. Each such vertex is a neighbor of some unmatched white vertex, so its $\text{tier}^\star$ is equal to r. Then, one by one, we pop a vertex $b \in B(G)$ and try to match it using SEARCH on all its white neighbors $w \in \mathcal{N}(b)$ such that $w.\,\text{rank}^* \leqslant r$. If a witness is found, we match b, otherwise its tier is at least $(r+1)$ and so it is again push back on the queue. After we complete one pass we are certain that all the vertices in the queue are of tier $(r+1)$ or higher, so we increase r and repeat this process as long as $r \leqslant \sqrt{2|V|}$. The vertices which are left in the queue at the end are the vertices of infinite tier. To easily differentiate between passes, we use two queues Q_1 and Q_2 and swap their contents when Q_1 becomes empty. See Algorithm 5 for the complete code listing.

Algorithm 5 Mimicking the behavior of the Hopcroft-Karp algorithm

```
1: procedure MatchLikeHK
2:     Initialize two FIFO queues Q1 ← B(G) and Q2 ← ∅
3:     for r from 1 to √(2|V|) do
4:         for each b ∈ Q1 do
5:             b ← Q1.pop
6:             if not MatchUntil(b, r) then
7:                 Q2.push(b)
8:         swap Q1 and Q2
```

Observe that Lemma 3.29 and Corollary 3.32, which claim respectively that the tiers are non-decreasing and that they bound from below the lengths of the shortest augmenting paths, are actually the only results necessary to derive the running time of Algorithm 5. More precisely, after r rounds we know that the smallest rank of any unmatched black vertex is at least r, and thus no augmenting path is shorter than $2r$. Therefore,

there can be at most $\frac{|V|}{2r}$ augmenting paths left, so we can spend $\mathcal{O}(|E|)$ on finding each of them. We get that the running time is bounded by $\mathcal{O}(r \cdot |E| + \frac{|V|}{2r} \cdot |E|)$, which simplifies to $\mathcal{O}(|E| \cdot |V|^{1/2})$ for $r = |V|^{1/2}$.

In other words, Algorithm 5 is an alternative offline matching algorithm with the same running time as the Hopcroft-Karp algorithm, but unlike the latter, it does not rely on disjoint shortest paths. It is also much simpler than the online version—we need only the basic properties of ranks and tiers. Moreover, similarly to the Hopcroft-Karp algorithm, we could stop Algorithm 5 after ε^{-1} rounds, for any $\varepsilon > 0$. By Observation 1.4 that gives us $(1-\varepsilon)$-approximation in $\mathcal{O}(|E| \cdot \varepsilon^{-1})$ time.

A slower, yet very interesting matching algorithm was given by Demange, Gale and Sotomayor [28]. It is based on an auction where each white vertex $w \in W(G)$ has some price p_w and black vertices bid for them. Each time vertex $b \in B(G)$ finds itself unmatched, it picks a white vertex w with the lowest price and bids $p_w + \delta$ for it—this way b takes w for itself, at the same time increasing its price. The authors show that after the prices reach bidders' valuations, this process reaches an equilibrium M such that $|V| \cdot \delta + |M| \geqslant |M'|$ for any other matching M'. Because in the unweighted case all the valuations are equal to 1, setting $\delta = (|V|+1)^{-1}$ yields a maximum matching algorithm that works in $\mathcal{O}(|E| \cdot |V|)$ time.

Observe, that the prices in that algorithm have similar function to $\text{rank}^\star$'s in our algorithm. In fact, what the rank-based algorithm does can be viewed as a very specific order of bids and rebids that form augmenting paths. In this light, a natural question arises—is it possible to apply the theorems of Chapter 3 to the algorithm of Demange, Gale and Sotomayor? Although it might be possible, the unconstrained nature of the auction makes it far from straightforward. In particular, our reasoning depends strongly on augmenting paths, however, the chains of bids may not reach any unmatched white vertex for many rounds. Still, in the offline setting we can improve the running time of their algorithm via different method. More precisely, adopting a hybrid approach that first calculates $(1-|V|^{-1/2})$-approximation using $\delta = (|V|+1)^{-1/2}$, and then proceeds with standard DFS-based augmentations, we can obtain $\mathcal{O}(|E| \cdot |V|^{1/2})$ running time.

Nevertheless, thanks to Theorems 3.16 and 3.36, by limiting ourselves to augmenting paths, we can achieve the same bounds with much simpler procedure. In particular, we can just treat the offline problem as a special case of the online setting and use Algorithm 3 which gives the same asymptotic running time as the approaches presented above.

Algorithm 6 Simple exact algorithm

```
1: procedure MatchExact
2:     for each b ∈ B(G) do
3:         MatchUntil(b, 2 · |M|^{1/2})
```

Naturally, the same applies to Algorithm 4.

Algorithm 7 Simple approximation algorithm

```
1: procedure MatchApprox(ε)
2:     for each b ∈ B(G) do
3:         MatchUntil(b, ε^{-1})
```

Neither of these procedures rely on the order in which vertices of $B(G)$ are processed, for example we can process them in random order. In opposition to the online setting, we were unable to find examples that would provide tight lower bounds. Especially, for the non-deterministic offline algorithm that picks a random permutation of black vertices and proceeds to match them in that particular order (see Algorithm 8). Practical experiments on many different classes of graphs failed to produce sufficiently high ranks to believe that Theorem 3.16 is tight in such case. We conjecture, that the expected maximum rank produced by Algorithm 8 is asymptotically strictly smaller than $|V|^{1/2}$, i.e., that it is asymptotically faster than the Hopcroft-Karp algorithm.

Algorithm 8 Exact randomized algorithm

```
1: procedure MatchExactRnd
2:     Set σ to be a random permutation of B(G).
3:     for each b ∈ σ do
4:         MatchUntil(b, 2 · |M|^{1/2})
```

Conjecture 4.7. *The Algorithm 8 runs in* $O\left(|E| \cdot |V|^{\alpha}\right)$ *time for some* $\alpha < \frac{1}{2}$.

4.4 Decremental Case

In this section we introduce a setting, where the vertices of a bipartite graph are being removed instead of inserted. We then present two decremental algorithms for this problem: an exact and an approximate one.

Consider an online problem where at the beginning we are given a bipartite graph $G^d = \langle W^d \uplus B^d, E^d \rangle$, and the vertices of W^d are removed from G^d in an online manner. The goal is to maintain during this process the maximum size matching in G^d.

Similarly to Definition 1.9, the input is a sequence of graphs $G_t^d = \langle W_t^d \uplus B, E_t^d \rangle$ such that

- $G_0^d = \langle W_0^d \uplus B^d, E_0^d \rangle = \langle W^d \uplus B^d, E^d \rangle$,
- $G_t^d = \langle W_t^d \uplus B^d, E_t^d \rangle = \langle W_{t-1} \setminus \{w_t\} \uplus B^d,\ E_{t-1}^d \setminus E_{w_t}^d \rangle$ where

$$E_{w_t}^d = E^d \cap \left\{ \{w_t, b\} \;\middle|\; b \in B \right\},$$

- $G_n^d = \langle W_n^d \uplus B^d, E_n \rangle = \langle \varnothing \uplus B^d, \varnothing \rangle$.

As before, the output is a sequence of matchings $M_0^d, M_1^d, \ldots, M_n^d$ such that M_t^d is a matching in G_t^d, which depends only on graphs $G_0^d, G_1^d, \ldots, G_t^d$.

Note an important difference with respect to the incremental problem. Here, we are removing white vertices $w_t \in W$, whereas previously we were adding black vertices $b_t \in B$. This change in colors will be useful when reducing the decremental problem to the incremental one.

We initialize the incremental algorithm with graph G^d, i.e., $G_0 = G^d$, and set M_0 to be the maximum cardinality matching in that graph, which we calculate in $\mathcal{O}\left(|E| \cdot |V|^{1/2}\right)$ time using $|B^d|$ turns of the incremental algorithm. When the vertex w_t should be removed from G^d, we add a new vertex b'_t together with an edge $\{b'_t, w'\}$. Denote by G_t the sequence of graphs resulting from this incremental process. We observe the following lemma.

Lemma 4.8. *For maximum matchings M_t in G_t and M_t^d in G_t^d we have*

$$|M_t| - t = |M_t^d|.$$

Proof. Consider matching $M_t' = M_t^d \cup \{\{b_i', w_i\} \mid 1 \leqslant i \leqslant t\}$. Certainly $|M_t'| - t = |M_t^d|$. If M_t' is not a maximum size matching in G_t then there exists an M_t'-augmenting path A. No such augmenting path can pass via any of the edges in $\{\{b_i', w_i\} \mid 1 \leqslant i \leqslant t\}$, because b_i' is of degree $\deg_{G_t}(b_i') = 1$. Hence, A would have to be an M_t^d-augmenting path in G_t^d what contradicts the maximality of M_t^d. Therefore, $|M_t| = |M_t'| = |M_t^d| + t$. □

Observe that it is straightforward to calculate M_t^d from M_t, namely it is enough to remove from M_t edges incident to w_i for $1 \leqslant i \leqslant t$. Combining these results with Theorem 4.3, which affirms the existence of the incremental algorithm working in $\mathcal{O}(|E| \cdot |V|^{1/2})$, we obtain the following corollary.

Corollary 4.9. There exists a decremental algorithm for the maximum bipartite matching problem that works in $\mathcal{O}(|E| \cdot |V|^{1/2})$ total time. □

We now prove that the same reduction works in the approximate case. In the incremental case we were using Observation 1.4 that relates the size of some matching M to the length of the shortest M-augmenting paths. 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. Nevertheless, it is possible to prove the guarantee on approximation ratio directly from the nonexistence of short augmenting paths.

Lemma 4.10. *Let M_t^d be the maximum cardinality matching in G_t^d and let M_t be any matching in G_t such the shortest M_t-augmenting path has length at least k for some $k \geqslant 3$. Then*

$$|M_t| - t \geqslant |M_t^d| \left(1 - \frac{2}{k-2}\right).$$

Proof. To bridge the gap between M_t and M_t^d we introduce two new matchings N_t and N_t^d defined as follows:

$$N_t = M_t \cap E_t^d \cup \left\{\{w_i, b_i'\} \;\middle|\; 1 \leqslant i \leqslant t\right\},$$
$$N_t^d = M_t \cap E_t^d = N_t \cap E_t^d.$$

Because the matching M_t does not admit augmenting paths of length 1, each vertex w_i has to be matched for $1 \leqslant i \leqslant t$. Therefore, N_t could be thought of as constructed from M_t by rematching vertices w_i to b_i' for $1 \leqslant i \leqslant t$, in particular $|N_t| = |M_t|$. On the other hand, N_t^d is M_t constrained to G_t^d, which also happens to be N_t constrained to G_t^d. For this reason $|N_t^d| = |M_t| - t$.

Suppose that A is an N_t-augmenting path of length l. Note that A does not contain any of the $\{w_i, b_i'\}$ edges, thus it is either an M_t-augmenting path, or one of its endpoints happens to be in $b_{w_i}^t$, a vertex matched in M_t to w_i for some $1 \leqslant i \leqslant t$. In such case, the path $A' = A \circ \left[b_{w_i}^t \leftrightarrow w_i \leftrightarrow b_i'\right]$ is an M_t-augmenting path of length $(l+2)$. This implies that the shortest N_t-augmenting path has length at least $(k-2)$.

Furthermore, because G_t^d is a subgraph of G_t and the vertices unmatched by N_t^d in G_t^d are precisely the N_t-free vertices of G_t, any N_t^d-augmenting path in G_t^d is necessarily an N_t-augmenting path in G_t. Hence, there are no shorter N_t^d-augmenting paths in G_t^d than $k-2$. In other words, due to Observation 1.4, $|N_t^d|$ is a $(1 - \frac{2}{k-2})$-approximation of $|M_t^d|$, and so

$$|M_t| - t = |N_t^d| \geqslant |M_t^d| \left(1 - \frac{2}{k-2}\right). \qquad \square$$

Although in the exact case we had to use $\mathcal{O}\left(|E| \cdot |V|^{1/2}\right)$ initialization to calculate M_0, in the approximate setting we can use Algorithm 7 which works in $\mathcal{O}\left(|E| \cdot \varepsilon^{-1}\right)$ time. Hence, by using the above lemma together with Theorem 4.5 we obtain an observation corresponding to Corollary 4.9.

Corollary 4.11. For any $\varepsilon > 0$, there exists a decremental algorithm that maintains $(1-\varepsilon)$-approximate maximum bipartite matching in $\mathcal{O}\left(|E| \cdot \varepsilon^{-1}\right)$

total time. □

Finally, we note that this reduction can be easily combined together with claims given in the Section 4.2 to obtain the following two corollaries.

Corollary 4.12. There exists a decremental algorithm for the maximum bipartite matching problem that finds augmenting paths of total length bounded by $\mathcal{O}(|V|^{3/2})$. □

Corollary 4.13. There exists a decremental algorithm that maintains $(1-\varepsilon)$-approximate matching in a bipartite graph and finds augmenting paths of total length bounded by $\mathcal{O}(|V| \cdot \varepsilon^{-1})$. □

4.5 Weighted Case

The maximum weight matching problem is a natural generalization of the task of finding a maximum cardinality matching. Unsurprisingly, the online bipartite matching problem with augmentations has its own weighted version, which will be the subject of this short section. The main idea is to reduce the weighted case to the maximum cardinality matching using the unfolded graph technique of Kao, Lam, Sung and Ting introduced in [64], which we recall below.

Let $G_w = \langle V_w, E_w \rangle$ be a graph where edge weights are given by a function $\text{weight} : E \to \{1, 2, \ldots, \mathcal{W}_{\max}\}$ for some natural number $\mathcal{W}_{\max}$ that represents the maximum weight of any edge. The *unfolded graph* $G = \langle V, E \rangle$ of G_w is defined as follows:

$$V = \left\{ v_i \;\middle|\; v \in V_w, i \in \{1, 2, \ldots, \mathcal{W}_{\max}\} \right\},$$
$$E = \left\{ \{u_i, v_{\mathcal{W}-i+1}\} \;\middle|\; \{u, v\} \in E_w,\ \mathcal{W} = \text{weight}(\{u, v\}),\ i \in \{1, 2, \ldots, \mathcal{W}\} \right\},$$

We have the following lemma.

Lemma 4.14 (Lemma 4.1 of [64]). *The weight of the maximum weight matching in G_w is equal to the size of the maximum cardinality 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—each time a new vertex b of G_w arrives, we just add the corresponding vertices $b_1, b_2, \ldots, b_{\mathcal{W}}$ for $\mathcal{W} = \max_{u \in \mathcal{N}(G_w)} \text{weight}(\{b, u\})$. We do not need to add vertices b_i for $i > \mathcal{W}$, because they are isolated, hence, irrelevant. Observe that a similarly property holds also for the white vertices of G, and as isolated vertices are unreachable, we do not have to represent them explicitly in any kind of structure. That means the algorithm does not need to know $\mathcal{W}_{\max}$ from the start. The unfolded graph G has at most $\mathcal{W}_{\max} \cdot |E_w|$ edges and $\mathcal{W}_{\max} \cdot |V_w|$ vertices, therefore, the following corollaries are immediate.

Corollary 4.15. There exist incremental and decremental algorithms that maintain the weight of the maximum weight bipartite matching in G_w which work in $\mathcal{O}\left(\mathcal{W}_{\max}^{3/2} \cdot |E_w| \cdot |V_w|^{1/2}\right)$ total time. □

Corollary 4.16. There exist incremental and decremental algorithms that maintain $(1 - \varepsilon)$-approximation of the weight of the maximum weight bipartite matching in G_w that work in $\mathcal{O}\left(\mathcal{W}_{\max} \cdot |E_w| \cdot \varepsilon^{-1}\right)$ total time. □

Chapter 5

Lower Bounds on the Ranks and Tiers

When developing the theory of ranks and tiers, we have bounded them from above by $\mathcal{O}(|M|^{1/2})$. In this chapter we provide graph instances which cause the ranks to be of order $\Omega(|M|^{1/2})$. This implies that all the presented upper bounds, including these from Theorem 3.16 and Theorem 4.3 are asymptotically tight.

Although there is only one place in Algorithm 1 where the ranks are increased, namely line 2, intuitively there are two possible reasons—due to an update, because of failed search, or due to application of an augmenting path, when the search has ended successfully. Therefore, we introduce two kinds of examples, in Section 5.1 and Section 5.2 respectively.

The first and easier kind of instances targets the searching process. The examples are tailored to specific graph traversal algorithms, and induce a high complexity due to the most common reason—because the search phase of almost any augmenting-path algorithm is expensive. It is well-known that the total length of all augmenting paths in the Hopcroft-Karp algorithm is $\mathcal{O}(|V| \log |V|)$, yet, it still needs $\mathcal{O}(|E| \cdot |V|^{1/2})$ time because of the search. In similar vein, the first kind of examples cause the algorithm to perform costly graph exploration in areas that produce no augmenting paths.

The second kind of examples targets the augmenting paths. With a carefully crafted graph instances we show that it is possible for the algorithm to produce augmenting paths of total length $\Omega(|V|^{3/2})$. As the construction is complex, we start with a basic building block called a *comb*, then an intermediate instance of size $\mathcal{O}(r^3)$, and finally a compressed graph of size $\mathcal{O}(r^2)$.

Note that although we prove that bounds in Theorem 3.16 and Theorem 4.3 are tight, it does not contradict Conjecture 4.7—all the examples below depend on the online setting, that is, the vertices are added in a specific order.

5.1 Examples Targeting the Searching Procedure

The examples we introduce in this section cause the search algorithm to explore the graph in a costly manner. For a fixed parameter r we construct a graph of size $\mathcal{O}(r^2)$, in which during each of the final r turns the ranks are increased $\Omega(r^2)$ times.

One could wonder, why then we could not use that technique to make each of the $|B|$ rounds costly. This is because in the initial stages the augmenting paths are, on average, too short. Moreover, each time the algorithm errs and picks the wrong direction, the ranks reflect the structure of the graph better and better. In other words, for the search procedure to get lost in the graph, we have to make it, in a sense, into a labyrinth. Yet, each time the algorithm strays away, it learns the structure of the maze. The $r = |V|^{1/2}$ value balances the average length and the number of the complicated augmenting paths, or, more intuitively, the need to make the graph relatively complex and the time it takes for the ranks to resemble its structure.

5.1.1 An Example for the Breadth-First Search

We start with an example for a BFS-based algorithm, because it is easy to force the algorithm to explore large portions of the graph—it is enough to make the augmenting paths all of the same length.

The graph instance consists of r disjoint paths, each of r black and $(r+1)$ white vertices. Observe, that there are r seeds, i.e., unpaired white vertices. As we consider the worst-case behavior, without lost of generality we can assume that they are all located in one of the endpoints. To complete the graph we add additional r black vertices, which we use to connect the paths together. More precisely, the (r^2+i)'th black vertex connects to the *matched* endpoint of paths $i, i+1, \ldots, r$. Figure 5.1 depicts the construction for $r = 6$ during turn $t = (r^2+3)$, the new black vertex is marked by an additional black circle.

Note, how the search, when it reached the black vertex denoted by b, first explored the descendants of b, rather than other vertices. This is because its tier according to rank^*'s was 2, which is strictly less than the tier of any other vertex. That didn't happen earlier, because before reaching b the available branches were equivalent, all of tier 3.

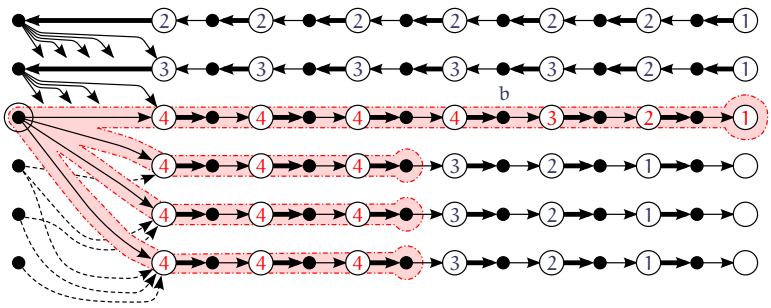

Figure 5.1: A graph instance that provides an $\Omega\left(|V|^{1/2}\right)$ lower bound for the BFS-based algorithm, realized for $r = 6$.

After the last r black vertices have been added, the highest rank^* reaches the value of $r+1 = 7$, while the sum of all the ranks can be estimated using the square pyramidal numbers, that is,

$$(r+1) + \sum_{w \in W} w.\,\text{rank}^* = \frac{2(r+1)^3 + 3(r+1)^2 + (r+1)}{6} \in \Theta(r^3).$$

5.1.2 An Example for the Depth-First Search

The graph instance, that causes the DFS-based algorithm to get lost, is just a modification of the example shown in the previous section. To force the DFS to traverse the graph instead of going straight for the seeds, we make it follow a path that results in the algorithm blocking itself due to vertices being marked visited. This ensures the algorithm has to explore all the other parts of the graph first, and only then it can retract its steps and reach the seed.

To construct the example for a given parameter r we take r disjoint paths of $(r+2)$ white vertices and $(r+1)$ black vertices each, and join them together with extra $(r \cdot r)$ edges that form zigzags perpendicular to these paths as shown in Figure 5.2. We then add a single vertex b that connects to paths' endpoints, similar to the (r^2+1)'th black vertex of the previous example. However, we additionally make b adjacent to a white vertex w, which will act as a relay and forward the depth-first searches to b. Finally, we insert the last r black vertices, all adjacent to the first path, sequentially, starting from the second matched vertex.

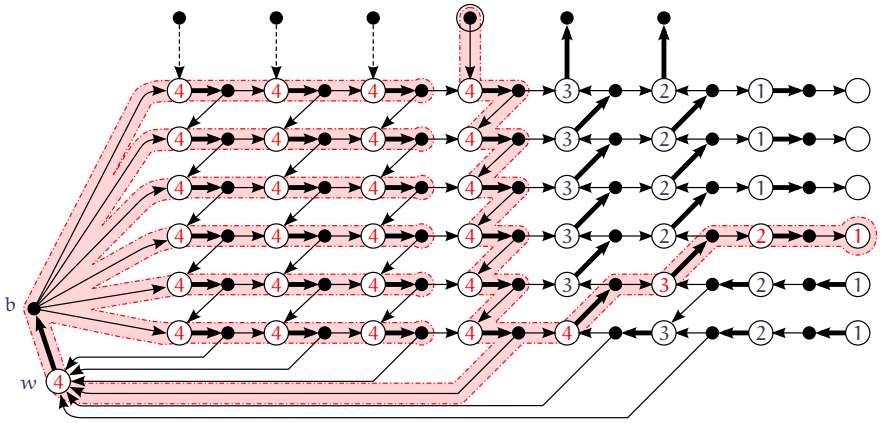

Figure 5.2: A graph instance providing an $\Omega\left(|V|^{1/2}\right)$ worst-case lower bound for the DFS-based procedure realized for $r = 6$.

The diagram above shows turn $t = r \cdot (r+1) + 1 + 3$, in which the third of the last r vertices is added. The red hue traces the DFS search tree:

it first goes through the zigzag to w and b and blocks its path to the seeds; after exploring the whole area on the left, it retraces its steps back to the zigzag and then follows to the right, finally reaching a seed.

Observation 5.1. For any $r \geqslant 2$ the described instance has $\Theta(r^2)$ vertices and $\Theta(r^2)$ edges. Moreover, the total sum of the ranks achieved by the Algorithm 3 in the worst case is of cubic order,

$$\sum_{w \in W} w.\mathrm{rank}^* \in \Omega(r^3). \qquad \square$$

Corollary 5.2. The bound in Theorem 4.3 is tight. $\square$

5.1.3 An Example for the Randomized Depth-First Search

One significant drawback of the example from the previous section is that it assumes the depth-first search uses graph edges in particular order. However, it is a common practice to permute the edges of an input graph prior to searching, precisely to avoid some worst-case behaviors of algorithms. In this section we transform the last example into a graph instance which gives a slightly worse bounds, but still works if the search considers the edges in a random order. More specifically, the construction is of size $\mathcal{O}(r^2 \log r)$ for a given parameter r and it produces ranks of total sum $\Omega(r^3)$ with a constant probability.

The key is to trick the algorithm to follow the zigzag rather than the path to the seed. To this end, instead of a single edge leading to the next path we use k such edges, as shown in Figure 5.3.

Assuming the algorithm choices at each vertex are independent, we can use $k \in \Theta(\log r)$ to ensure the algorithm will use all the zigzags with a constant probability. More precisely, there are r zigzags each of length r, thus we can set $k = 1 + 2\lceil \log_2 r \rceil$ to ensure that the algorithm fails to follow all the zigzags with probability at most $r \cdot r \cdot 2^{-k} = 1/2$.

Finally, the graph has to be constructed online, taking into account algorithm's decisions, to ensure the paths are appropriately directed. We start with the rallying vertex w which is determined by the first matched edge of b. For technical reasons, there is another pair w' and b', which

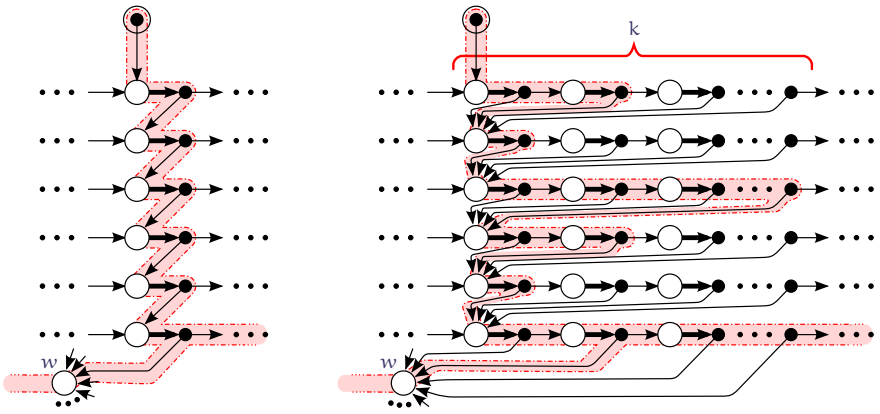

Figure 5.3: Modification to accomodate for randomized edge order: a part of the original graph for $r = 6$ (left) and the transformed instance (right).

allows b' to connect w to the endpoints of the paths after all these paths have been constructed. Immediately after b has been added, we insert the original black vertices of the zigzags, starting with the ones closest to w. The newly created matched edges determine which white vertices are going to fulfill the role of the original white vertices of zigzags. Then we add all the rest, from right to left, so that the only available path of rank zero is to their left. The aforementioned vertex b' is added as the $(r+1)$'th vertex from the end, which at that turn has no other option, but to match to w'. That creates the whole structure of the graph, ready for the last r vertices, added from right to left, what increases the ranks to $(r+1)$ with probability $1/2$.

One way to subvert this type of graph instance is to randomize the search with regard to vertices, for example, each time a white vertex increases its rank we assign it a new random number from interval $[0, 1]$. We use these numbers to maintain a priority queue which indicates in which direction the algorithm should explore next. Despite the additional overhead of the priority queue, practical experiments show that such an algorithm performs on adversarially constructed instances slightly faster than other versions. This leads us to believe that its expected rank is actually smaller by a polylog factor than in the worst-case.

5.2 Examples Targeting the Augmenting Paths

This section describes a construction of another graph that lower-bounds the ranks by $\Omega(|V|^{1/2})$ in the worst case. The difference is that it also makes the algorithm to apply augmenting paths of total length bounded from below by $\Omega(|V|^{3/2})$.

As the structure is not easy to characterize, we first introduce a basic building block called a *comb*. Then, we follow with a simple example of cubic order, i.e., for a given parameter r we will construct an instance of size $\mathcal{O}(r^3)$ that implies ranks linear in r. In other words, it gives a $\Omega(|V|^{1/3})$ lower bound on the ranks, along with $\Omega(|V|^{4/3})$ on the total length of augmenting paths. Finally, we describe how to condense it to obtain a square order example, that is, an instance of size $\mathcal{O}(r^2)$ achieving ranks in $\Omega(r)$, and argue why such compression is possible.

5.2.1 The Comb

The basic building block is a structure we call an $\langle \alpha, \beta \rangle$-comb. It consists of a path of length 2α that joins α disjoint components of rank β, that is, any structures or subgraphs that allow for α pairwise vertex-disjoint seeded paths, each of rank β.

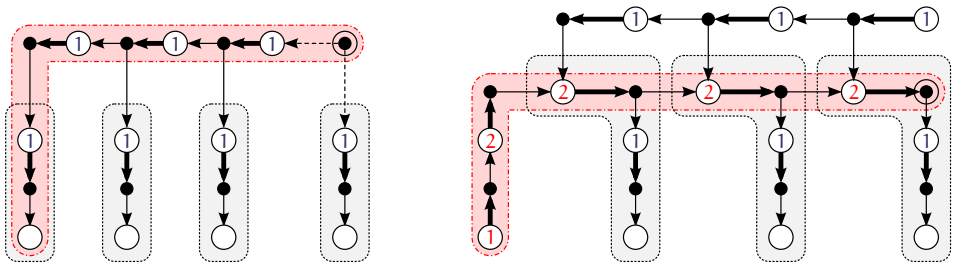

Figure 5.4: The left diagram depicts a $\langle 3, 1 \rangle$-comb with an additional component of rank 1, the circled black vertex is the one to be added. The right side shows two combs, $\langle 3, 1 \rangle$ (lower) and $\langle 3, 2 \rangle$ (upper) stacked together, to present how combs of higher rank can reuse the smaller structures.

The combs usually start in skim form, that is, their long path is of rank 1. By applying components of increasing rank on both sides, an $\langle \alpha, \beta \rangle$-comb can be fattened up to $\beta + 1$. This is possible, because the long path is the lowest-ranked path to some seed until the final phase, when it is of the same rank, namely β, and thus still feasible for the algorithm to use. Such a fattened comb is a great source of $(\beta + 1)$-ranked components for another structure, that could be stacked just above it, as presented in Figure 5.4. This construction leads to the following example.

5.2.2 The Instance of Cubic Order

Consider r combs stacked on top of each other. Suppose that the top one is a $\langle 2, r \rangle$-comb. Then, the next one has to be of length 2 and supply an additional component to fatten up the first, i.e., a $\langle 3, r-1 \rangle$-comb. The third from the top has to support the three paths from the previous comb and accommodate further two components, that is, we need a $\langle 5, r-2 \rangle$-comb. Following the suit, the general rule for the i-th comb is $\left\langle \frac{1}{2}(i^2 + i + 4), r - i \right\rangle$.

Figure 5.5 presents the graph for $r = 6$. The numbers next to the black vertices indicate the insertion order, while the numbers near the white vertices denote the corresponding pairs matched by the augmenting path. To avoid confusion, the white vertices which are not seeds were marked with a cross instead of the usual non-zero rank*. The special inverted-color vertex belonging to the top comb is the one that at the end reaches the highest rank. Note that after the addition of all the 22 black vertices, there will be still some seeds left—we do not saturate all of them to make the structure of the graph simpler and more apparent.

Augmenting paths traverse all the horizontal edges of the combs in bottom-up manner: one path for the bottommost comb, two for the next, and so on, up to r augmenting paths for the topmost $\langle 2, r \rangle$-comb. Such paths are tiered, because of two reasons. First, the edges between combs point downward, i.e., it is not possible to reach upper combs. Second, the tiers of vertices of any comb in the fattening phase are smaller or equal than the tiers of vertices of combs below. More specifically, for $\langle \alpha, \beta \rangle$-comb,

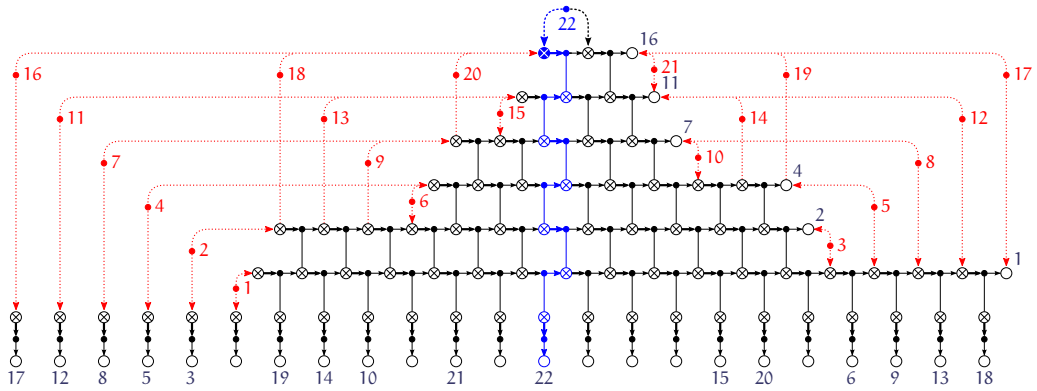

Figure 5.5: The graph of cubic order for $r = 6$. The numbers at black vertices denote the insertion order, while the numbers at white vertices indicate the seed that was matched at that turn. The red dotted lines denote edges that will be added with an appropriate black vertex. The inverted-color vertex at the top of the graph is the one to achieve the highest rank after addition of all the 22 black vertices.

the first $(\beta - 1)$ indicated paths are of strictly lower rank, while the last path is of equal rank. It means that, even if some particular run might not follow these exact paths, in the worst case the algorithm could use them.

5.2.3 The Compressed Graph

The issue with the previous instance is that we require multiple components for each of the stacked combs to make them fat. However, we may observe that we are doing the same job many times, that is, any $\langle \alpha, \beta \rangle$-comb has to be fattened from k to $k+1$ for any $k \leqslant \beta$.

To avoid this inefficiency we are going to improve the previous example, so that a single component of rank k is enough to make all the appropriate combs into $k+1$. To achieve it, we join all the combs in a sequential manner so that a single, long augmenting path could go through them all, as in the sketch below. Nevertheless, to allow for subgraph reuse, they are at the same time stacked exactly as in the cubic example. This modification is the base of our compression, and it allows us to construct an example implying $\Omega(|V|^{1/2})$ lower bound on the maximum rank using only the augmenting paths.

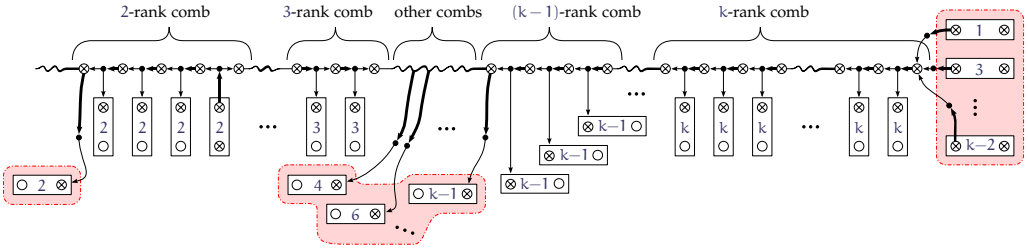

Figure 5.6: Combs in a series configuration. The additional components are marked by a light red background, the last two were of rank $(k-2)$ and $(k-1)$.

To obtain a structure similar to the one in the previous paragraph, we apply the components only to appropriate combs by joining them in the middle of the chain. In the diagrams k is odd, which means that after the k-th pass (see Figure 5.7), the path in the graph leads from left to right (similarly for 3-rank comb), while the part of paths near combs of even rank goes from right to left, as for 2-rank and $(k-1)$-rank combs.

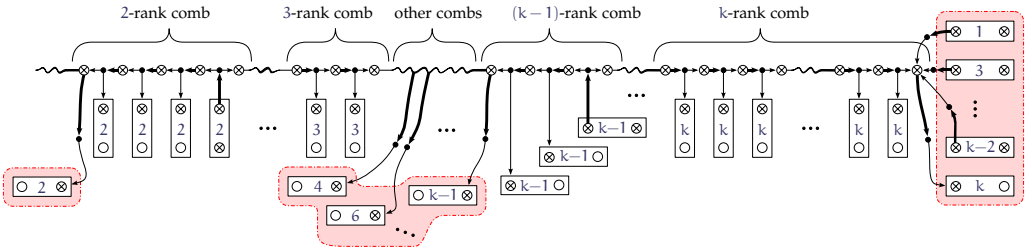

Figure 5.7: The structure after the component of rank k has been added.

The whole concrete example might look as shown in Figure 5.8, realized for $r = 5$. The green dashed edges are those which stack two combs on top of each other, as in the cubic case. Observe that 4-rank comb does not need additional backing by components of rank 2, as the 3-rank comb provides it when it is necessary, i.e., at that time these 3-rank components are of rank 2. The red dotted edges are responsible for making the combs fat, and the blue inverted-color vertex is the one achieving the maximum rank.

The last five augmenting paths go as noted in the list below. Figure 5.9 presents the state just before the last path is applied.

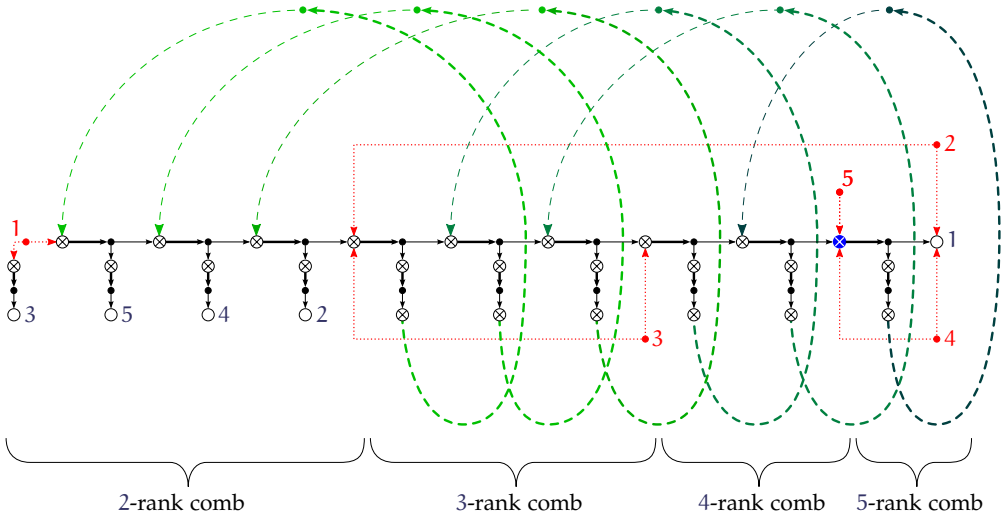

Figure 5.8: The example of square order for $r = 5$. State in which the five final seeds are still unmatched.

1. From left-most vertex marked by number 1, straight right, by the black solid path, until its white counterpart.

2. Starting near 2 on the red dotted arc, down onto the black solid path, and then left until junction leading to white 2, where it finishes.

3. From red 3 vertex, one segment up, then right until the end, via red dotted arc and vertex 2 back onto black solid path, one section right, and through green dashed arc, finally reaching white vertex labeled by number 3.

4. Up to white 1 and left, through the blue inverted-color vertex, then further left, until the first junction down which would lead by two green dashed arcs and small number of black edges to the white pair of 4.

5. Down to the inverted-color vertex, then right once and down, three hops via green dashed arcs, and finally left and down again to the white 5.

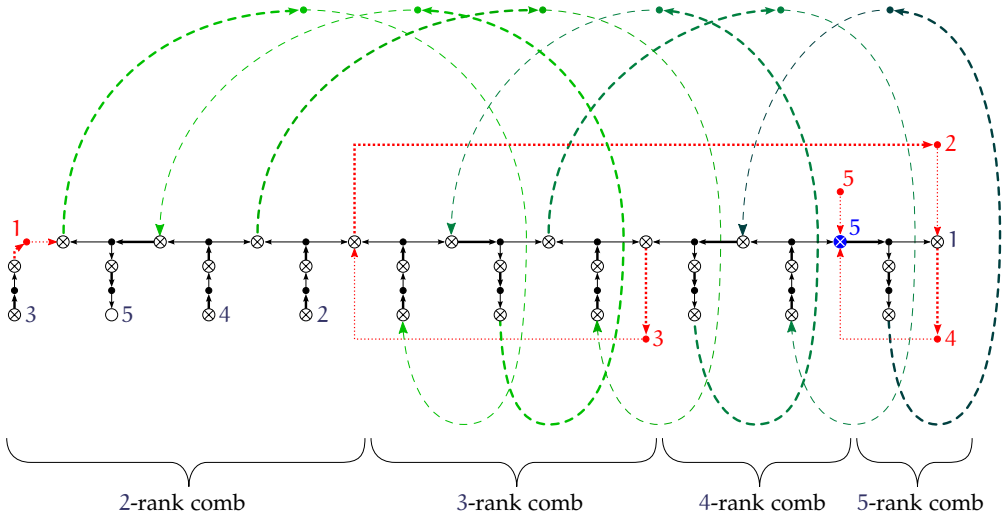

Figure 5.9: A single seed is left, the highest ranked edge is of rank 5.

The boundaries between combs are blurry, because of frequent edge orientation changes. However, we can still count the number of white vertices, and so, the 5-rank comb uses 3, 4-rank comb uses $2 \cdot 3$, the next one $3 \cdot 3$ and the last, 2-rank comb uses $4 \cdot 3$. Generalizing for arbitrary r, we have that k-rank comb needs $(r - k + 1) \cdot 3$ nodes and the sum $3 \sum_{k=2}^{r} r - k + 1 = 3 \sum_{k=1}^{r-1} k = \frac{3}{2} \cdot r \cdot (r - 1)$ makes the quadratic nature of the considered example self-evident.

Corollary 5.3. The bound in Theorem 3.16 is tight.

Conclusions

In this thesis we have considered a resource allocation problem, namely the setting of online bipartite matching with augmentations. It can be summarized as follows. We are given a bipartite graph $G = \langle W \uplus B, E \rangle$, which is presented in a one-sided online fashion, that is, the white set W is known from the start, while the black set B is revealed gradually during the algorithm run. More precisely, each turn, a single black vertex $b \in B$ arrives, together with all its incident edges. Our main aim is to maintain the maximum cardinality matching, in particular, the decisions of the procedure are revocable—a matched vertex can be reassigned at a later time. Our secondary objective is to keep the number of these rematchings low.

This problem was introduced in 1995 by Grove, Kao, Krishnan and Vitter [53], and the results included in this thesis were the first non-trivial bounds for the general case.

In the meantime, Chaudhuri, Daskalakis, Kleinberg and Lin [24] analyzed a few special cases and stated a conjecture that the shortest augmenting path algorithm achieves $\mathcal{O}(|V| \log |V|)$ bound on the total length of augmenting paths.

We took that approach in Chapter 2, where we have proved that the total length of shortest augmenting paths does not exceed $\mathcal{O}(|V| \log^2 |V|)$ when the underlying graph is a tree. Nevertheless, since that bound does not match tightly the $\Omega(|V| \log |V|)$ lower bound, the question whether we can do better imposes itself. Observe that Lemmas 2.9 and 2.10 already satisfy the $\mathcal{O}(|V| \log |V|)$ upper bound, while the inequalities of Lemma 2.12 are coarse—not only we sum sizes of whole subtrees instead of just paths in them, but we assume the worst-case logarithmic usage for every single

subtree and thus multiply them all by another logarithmic factor. Furthermore, intuitively, a sequence of trees with the longest shortest augmenting paths should have the largest possible ratio of the total length of all final dispatching paths to the sum of lengths of all paths. However, this ratio seems to be the largest when all the black vertices but one are of degree 2, and the graph is just a long path. This intuition appears to work also for the general bipartite graphs and it agrees perfectly with the aforementioned conjecture of Chaudhuri, Daskalakis, Kleinberg and Lin from [24], which remains unproved.

Nevertheless, the proof of Theorem 2.6 happens to have a surprising characteristic—it does not rely on the matching it calculates, only on the structure of the underlying graph. That prompted us to consider the following game: each turn, after the algorithm calculates the new matching, an adversary changes that matching arbitrarily, preserving only its cardinality. The proof of Theorem 2.6 still holds within such game and thus the total length of all the augmenting paths is bounded from above by $\mathcal{O}(|V| \log^2 |V|)$, given that the final graph is a tree. Moreover, we were unable to find an $\omega(|V| \log |V|)$ example even for general bipartite graphs. Analysis of a number of cases led us to the concept of *minimum surplus* from Definition 1.10 and the following conjecture.

Conjecture 2.14. *The total length of all the shortest augmenting paths in the online bipartite matching problem with augmentations is bounded from above by* $\mathcal{O}(|V| \log |V|)$*, even if the adversary is allowed to change the calculated matching at every turn, in any way that preserves its cardinality.*

In Chapters 3 to 5 we have considered another heuristic for the online bipartite matching problem with augmentations—for each vertex $w \in W$ we kept track of how many augmenting paths have used w, and we tried to greedily minimize the maximum of these counters. This technique yielded an algorithm that produces augmenting paths of total length bounded from above by $\mathcal{O}(|V|^{3/2})$, while its running time is $\mathcal{O}(|E| \cdot |V|^{1/2})$. We also gave $(1 - \varepsilon)$-approximation algorithm that works in $\mathcal{O}(|E|\varepsilon^{-1})$ and generates paths of $\mathcal{O}(|V|\varepsilon^{-1})$ total length. Furthermore, we have extended these results to the decremental-only and weighted settings by using sim-

ple reductions. The bounds of the former are exactly the same as in the incremental-only instance, both for the exact and for the approximate algorithm. In the case of weighted graphs we have used the unfolded graph technique of Kao, Lam, Sung and Ting [64], and obtained running times of $\mathcal{O}\left(W^{3/2} \cdot |E| \cdot |V|^{1/2}\right)$ and $\mathcal{O}\left(W \cdot |E| \cdot \varepsilon^{-1}\right)$ for exact and $(1-\varepsilon)$-approximation algorithm respectively. Finally, in Chapter 5 we have demonstrated examples that provide tight lower bounds for Theorems 3.16 and 3.36.

It is worth mentioning that the designed procedure, Algorithm 3, has a few nice characteristics. For example, due to the rank minimization the algorithm spreads its work over the whole graph as much as possible. In particular, such behavior simplifies the analysis, because all the nodes in the graph have the same bounds. Moreover, it could help in a practical setting, e.g., parallelization, or allowing an approach similar to the one assumed by Blelloch, Fineman and Shun in the case of greedy sequential maximum matchings [17].

Furthermore, ranks and tiers provide the algorithm much more freedom than the shortest augmenting paths, while they carry enough structure to imply strong bounds. This way, not only we can avoid repeating bad decisions too often, but perhaps even overlay a finer path-seeking strategies, or at least use heuristics tuned to some specific practical applications. Note that Algorithm 3 is rather easy to modify and adjust—it has very few special cases and treats the vertices uniformly, which makes it an ideal candidate for a building block of more complex algorithms. Finally, the technique of ranks and tiers introduced in Chapter 3 is a simple combinatorial one, it appears applicable for a wide range of other settings, not only matchings.

Still, in this context the most straightforward generalization that comes to mind is whether the ranks and tiers would work for general, non-bipartite graphs, even in the offline case. Frequently over the years, each time a new result was proven for the offline bipartite matching problem, a corresponding result for the general graphs followed soon after. The biggest problem here is, as usual, an effective search for the augmenting paths. As the skew-symmetric graphs appear to be the easiest way to define ranks and tiers on general graphs, applying the methods used by

Goldberg and Karzanov in [50] seem to be an interesting line of research.

This points to the biggest limitation of ranks and tiers—the concepts seem to be tied to the one-sided online setting—if we were to allow insertions of vertices on the both sides of graph, a trivial example of a path would cause the ranks to be increased to $\Omega(|V|)$. Is this an intrinsic feature or maybe it is possible to modify the definitions to accommodate insertions on both sides?

Another open problem is whether the ranks and tiers technique can be applied to the weighted setting in a better way. Could we combine it with the scaling methods that have yielded $\mathcal{O}(|E| \cdot |V|^{1/2} \cdot \log(W \cdot |V|))$ time in [47] or $\mathcal{O}(|E| \cdot |V|^{1/2} \cdot \log W)$ time in [32]? Similar considerations apply also to vertex-weighted graphs and cases where the edge- and vertex-labels relate to capacities rather than just costs or benefits.

In the context of Section 3.4, it would be interesting to see whether the ranks and tiers technique would work for the maximum flow problem. It is certainly not a coincidence that for the maximum cardinality matching and the maximum unit flow problems we arrive at the same bounds as the algorithm of Dinitz [30]. Would the generalization of our procedure to general flows work in $\mathcal{O}(|V|^2 \cdot |E|)$? Even more intriguing are the parallels between the relaxed ranks in our algorithm and the labels in the preflow-based push-relabel maximum flow algorithm of Goldberg and Tarjan [52]. For example, the black vertices b_i for $i > t$ which have not yet arrived in turn t could be considered a positive excess in ${G_n}^{\bowtie M_t}$, while the set of tiered edges and the admissible network fulfill similar roles despite the former not being acyclic. Exploring these connections could improve our understanding of the matchings and flows.

Finally, the open problem which the author finds most intriguing is the one stated as Conjecture 4.7 at the end of Section 4.3 (see also below). Although it is not a rare occurrence that the algorithm's practical running times appears better than its analytic guarantees [33], and it is more that just possible that the classes of instances the Algorithm 8 was tested on simply did not contain the structures that cause the worst-case behavior, the procedure randomized in this way has certain characteristics that are not available in other approaches. For example, even if we were to

consider the vertices of $B(G)$ in the Hopcroft-Karp algorithm during each phase in a random order, we are still forced to pick those, which admit an augmenting path of minimum length. On the other hand, because of the online origin of Algorithm 8, the randomness provided by the input permutation has a distinctive rigid feel to it—impossible to remove or circumvent by the adversary, it influences the algorithm decisions in a way that was not possible before. Therefore, in the author's opinion settling Conjecture 4.7 either negatively or positively would be a significant result in algorithmic graph theory.

Conjecture 4.7. *If the vertices arrive in a random order, then the ranks and tiers technique yields an algorithm that runs in $\mathcal{O}(|E| \cdot |V|^{\alpha})$ time for some α strictly smaller than $1/2$.*

Bibliography

[1] A. Abboud and V. Vassilevska Williams, *Popular conjectures imply strong lower bounds for dynamic problems*, in FOCS 2014 [44], pp. 434–443.

[2] G. Aggarwal, G. Goel, C. Karande, and A. Mehta, *Online vertex-weighted bipartite matching and single-bid budgeted allocations*, in Randall [86], pp. 1253–1264.

[3] D. Alberts and M. R. Henzinger, *Average case analysis of dynamic graph algorithms*, in Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California., K. L. Clarkson, ed., ACM/SIAM, 1995, pp. 312–321.

[4] H. Alt, N. Blum, K. Mehlhorn, and M. Paul, *Computing a maximum cardinality matching in a bipartite graph in time* $\mathcal{O}\left(n^{1.5}\sqrt{m/\log n}\right)$, Information Processing Letters, 37 (1991), pp. 237–240.

[5] A. Anand, S. Baswana, M. Gupta, and S. Sen, *Maintaining approximate maximum weighted matching in fully dynamic graphs*, in Proceedings of the 32nd International Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India, D. D'Souza, T. Kavitha, and J. Radhakrishnan, eds., vol. 18 of LIPIcs, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2012, pp. 257–266.

[6] B. V. Ashwinkumar and R. Kleinberg, *Randomized online algorithms for the buyback problem*, in Leonardi [68], pp. 529–536.

[7] J. Aspnes, Y. Azar, A. Fiat, S. A. Plotkin, and O. Waarts, *On-line routing of virtual circuits with applications to load balancing and machine scheduling*, J. ACM, 44 (1997), pp. 486–504.

[8] Y. Azar, *On-line load balancing*, in Online Algorithms, The State of the Art, A. Fiat and G. J. Woeginger, eds., vol. 1442 of Lecture Notes in Computer Science, Springer, 1996, pp. 178–195.

[9] M. Babaioff, J. D. Hartline, and R. D. Kleinberg, *Selling ad campaigns: online algorithms with cancellations*, in Chuang et al. [25], pp. 61–70.

[10] S. Baswana, M. Gupta, and S. Sen, *Fully dynamic maximal matching in $O(\log n)$ update time*, in Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011, R. Ostrovsky, ed., IEEE Computer Society, 2011, pp. 383–392.

[11] ——, *Fully dynamic maximal matching in $O(\log n)$ update time*, SIAM J. Comput., 44 (2015), pp. 88–113.

[12] C. Berge, *Two theorems in graph theory*, Proceedings of the National Academy of Sciences of the United States of America, 43 (1957), pp. 842–844.

[13] P. Berman, M. Charikar, and M. Karpinski, *On-line load balancing for related machines*, J. Algorithms, 35 (2000), pp. 108–121.

[14] A. Bernstein and C. Stein, *Fully dynamic matching in bipartite graphs*, in Automata, Languages, and Programming (Proceedings of the 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Part I), M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, eds., vol. 9134 of Lecture Notes in Computer Science, Springer, 2015, pp. 167–179.

[15] S. Bhattacharya, M. Henzinger, and G. F. Italiano, *Deterministic fully dynamic data structures for vertex cover and matching*, in Indyk [61], pp. 785–804.

[16] B. E. Birnbaum and C. Mathieu, *On-line bipartite matching made simple*, SIGACT News, 39 (2008), pp. 80–87.

[17] G. E. Blelloch, J. T. Fineman, and J. Shun, *Greedy sequential maximal independent set and matching are parallel on average*, in Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2012, Pittsburgh, PA, USA, June 25-27, 2012, G. E. Blelloch and M. Herlihy, eds., ACM, 2012, pp. 308–317.

[18] B. Bosek, D. Leniowski, P. Sankowski, and A. Zych, *Online bipartite matching in offline time*, in FOCS 2014 [44], pp. 384–393.

[19] ——, *Shortest augmenting paths for online matchings on trees*, in Approximation and Online Algorithms (Proceedings of the 13th International Workshop, WAOA 2015, Patras, Greece, September 17-18, 2015), 2015.

[20] G. S. Brodal, L. Georgiadis, K. A. Hansen, and I. Katriel, *Dynamic matchings in convex bipartite graphs*, in Mathematical Foundations of Computer Science 2007 (Proceedings of the 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007), L. Kucera and A. Kucera, eds., vol. 4708 of Lecture Notes in Computer Science, Springer, 2007, pp. 406–417.

[21] N. Buchbinder, K. Jain, and J. Naor, *Online primal-dual algorithms for maximizing ad-auctions revenue*, in Algorithms – ESA 2007 (Proceedings of the 15th Annual European Symposium, ESA 2007, Eilat, Israel, October 8-10, 2007), L. Arge, M. Hoffmann, and E. Welzl, eds., vol. 4698 of Lecture Notes in Computer Science, Springer, 2007, pp. 253–264.

[22] J. Calmet and F. Ollivier, *Editors' foreword*, Applicable Algebra in Engineering, Communication and Computing, 20 (2009), pp. 1–4.

[23] M. Charikar, M. Henzinger, and H. L. Nguyen, *Online bipartite matching with decomposable weights*, in Algorithms – ESA 2014 (Proceedings of the 22th Annual European Symposium, ESA 2014, Wroclaw, Poland, September 8-10, 2014), A. S. Schulz and D. Wagner,

eds., vol. 8737 of Lecture Notes in Computer Science, Springer, 2014, pp. 260–271.

[24] K. Chaudhuri, C. Daskalakis, R. D. Kleinberg, and H. Lin, *Online bipartite perfect matching with augmentations*, in INFOCOM 2009 (Proceedings of the 28th IEEE International Conference on Computer Communications, INFOCOM 2009, 19-25 April 2009, Rio de Janeiro, Brazil), IEEE, 2009, pp. 1044–1052.

[25] J. Chuang, L. Fortnow, and P. Pu, eds., *Proceedings 10th ACM Conference on Electronic Commerce (EC-2009), Stanford, California, USA, July 6–10, 2009*, ACM, 2009.

[26] F. Constantin, J. Feldman, S. Muthukrishnan, and M. Pál, *An online mechanism for ad slot reservations with cancellations*, in Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009, C. Mathieu, ed., SIAM, 2009, pp. 1265–1274.

[27] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, McGraw-Hill Higher Education, 2nd ed., 2001.

[28] G. Demange, D. Gale, and M. Sotomayor, *Multi-item auctions*, Journal of Political Economy, 94 (1986), pp. pp. 863–872.

[29] N. R. Devenur and T. P. Hayes, *The adwords problem: online keyword matching with budgeted bidders under random permutations*, in Chuang et al. [25], pp. 71–78.

[30] Y. Dinitz, *Dinitz' algorithm: The original version and even's version*, in Theoretical Computer Science, O. Goldreich, A. L. Rosenberg, and A. L. Selman, eds., Springer-Verlag, Berlin, Heidelberg, 2006, pp. 218–240.

[31] R. Duan and S. Pettie, *Linear-time approximation for maximum weight matching*, J. ACM, 61 (2014), pp. 1:1–1:23.

[32] R. Duan and H. Su, *A scaling algorithm for maximum weight matching in bipartite graphs*, in Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012, Y. Rabani, ed., SIAM, 2012, pp. 1413–1424.

[33] I. S. Duff, K. Kaya, and B. Uçar, *Design, implementation, and analysis of maximum transversal algorithms*, ACM Trans. Math. Softw., 38 (2011), pp. 13:1–13:31.

[34] J. Edmonds, *Paths, trees, and flowers*, Canad. J. Math., 17 (1965), pp. 449–467.

[35] L. Epstein and A. Levin, *Robust algorithms for preemptive scheduling*, Algorithmica, 69 (2014), pp. 26–57.

[36] L. Epstein, A. Levin, D. Segev, and O. Weimann, *Improved bounds for online preemptive matching*, in Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 – March 2, 2013, Kiel, Germany, N. Portier and T. Wilke, eds., vol. 20 of LIPIcs, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2013, pp. 389–399.

[37] L. Epstein and J. Sgall, *A lower bound for on-line scheduling on uniformly related machines*, Oper. Res. Lett., 26 (2000), pp. 17–22.

[38] T. Feder and R. Motwani, *Clique partitions, graph compression, and speeding-up algorithms*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA, C. Koutsougeras and J. S. Vitter, eds., ACM, 1991, pp. 123–133.

[39] J. Feldman, M. Henzinger, N. Korula, V. S. Mirrokni, and C. Stein, *Online stochastic packing applied to display ad allocation*, in Algorithms – ESA 2010 (Proceedings of the 18th Annual European Symposium, ESA 2010, Liverpool, UK, September 6-8, 2010, Part I), M. de Berg and U. Meyer, eds., vol. 6346 of Lecture Notes in Computer Science, Springer, 2010, pp. 182–194.

[40] J. Feldman, N. Korula, V. S. Mirrokni, S. Muthukrishnan, and M. Pál, *Online ad assignment with free disposal*, in Leonardi [68], pp. 374–385.

[41] J. Feldman, A. Mehta, V. S. Mirrokni, and S. Muthukrishnan, *Online stochastic matching: Beating* $1 - 1/e$, in Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA, IEEE Computer Society, 2009, pp. 117–126.

[42] *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, IEEE Computer Society, 2012.

[43] *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, IEEE Computer Society, 2013.

[44] *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, IEEE Computer Society, 2014.

[45] L. Fortnow and S. P. Vadhan, eds., *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, ACM, 2011.

[46] H. N. Gabow and R. E. Tarjan, *A linear-time algorithm for a special case of disjoint set union*, Journal of Computer and System Sciences, 30 (1985), pp. 209–221.

[47] H. N. Gabow and R. E. Tarjan, *Faster scaling algorithms for network problems*, SIAM J. Comput., 18 (1989), pp. 1013–1036.

[48] G. Goel and A. Mehta, *Online budgeted matching in random input models with applications to adwords*, in Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008, S. Teng, ed., SIAM, 2008, pp. 982–991.

[49] G. Goel and P. Tripathi, *Matching with our eyes closed*, in FOCS 2012 [42], pp. 718–727.

[50] A. V. Goldberg and A. V. Karzanov, *Maximum skew-symmetric flows and matchings*, Math. Program., 100 (2004), pp. 537–568.

[51] A. V. Goldberg and R. Kennedy, *Global price updates help*, SIAM J. Discrete Math., 10 (1997), pp. 551–572.

[52] A. V. Goldberg and R. E. Tarjan, *A new approach to the maximum flow problem*, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA, J. Hartmanis, ed., ACM, 1986, pp. 136–146.

[53] E. F. Grove, M.-Y. Kao, P. Krishnan, and J. S. Vitter, *Online perfect matching and mobile computing*, in Algorithms and Data Structures (Proceedings of the 4th International Workshop, WADS '95, Kingston, Canada, August 16–18, 1995), 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.

[54] A. Gupta, A. Kumar, and C. Stein, *Maintaining assignments online: Matching, scheduling, and flows*, in Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, C. Chekuri, ed., SIAM, 2014, pp. 468–479.

[55] M. Gupta and R. Peng, *Fully dynamic* $(1+\varepsilon)$*-approximate matchings*, in FOCS 2013 [43], pp. 548–557.

[56] B. Haeupler, V. S. Mirrokni, and M. Zadimoghaddam, *Online stochastic weighted matching: Improved approximation algorithms*, in Internet and Network Economics (Proceedings of the 7th International Workshop, WINE 2011, Singapore, December 11-14, 2011), N. Chen, E. Elkind, and E. Koutsoupias, eds., vol. 7090 of Lecture Notes in Computer Science, Springer, 2011, pp. 170–181.

[57] P. Hall, *On representatives of subsets*, Journal of the London Mathematical Society, 10 (1935), pp. 26–30.

[58] F. L. Hitchcock, *The distribution of a product from several sources to numerous localities*, J. Math. Phys., 20 (1941), pp. 224–230.

[59] 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.

[60] O. H. Ibarra and S. Moran, *Deterministic and probabilistic algorithms for maximum bipartite matching via fast matrix multiplication*, Information Processing Letters, 13 (1981), pp. 12–15.

[61] P. Indyk, ed., *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, SIAM, 2015.

[62] Z. Ivkovic and E. L. Lloyd, *Fully dynamic maintenance of vertex cover*, in Graph-Theoretic Concepts in Computer Science (Proceedings of the 19th International Workshop, WG '93, Utrecht, The Netherlands, June 16-18, 1993), J. van Leeuwen, ed., vol. 790 of Lecture Notes in Computer Science, Springer, 1993, pp. 99–111.

[63] L. Kantorovich, *On the translocation of masses*, Doklady Akad. Nauk SSSR, 37 (1942), pp. 199–201.

[64] M. Kao, T. W. Lam, W. Sung, and H. Ting, *A decomposition theorem for maximum weight bipartite matchings with applications to evolutionary trees*, in Algorithms – ESA '99 (Proceedings of the 7th Annual European Symposium, ESA '99, Prague, Czech Republic, July 16-18, 1999), J. Nesetril, ed., vol. 1643 of Lecture Notes in Computer Science, Springer, 1999, pp. 438–449.

[65] C. Karande, A. Mehta, and P. Tripathi, *Online bipartite matching with unknown distributions*, in Fortnow and Vadhan [45], pp. 587–596.

[66] 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.

[67] N. Korula, V. S. Mirrokni, and M. Zadimoghaddam, *Bicriteria online matching: Maximizing weight and cardinality*, in Web and Internet Economics (Proceedings of the 9th International Conference, WINE 2013, Cambridge, MA, USA, December 11-14, 2013), Y. Chen and N. Immorlica, eds., vol. 8289 of Lecture Notes in Computer Science, Springer, 2013, pp. 305–318.

[68] S. Leonardi, ed., *Internet and Network Economics (Proceedings of the 5th International Workshop, WINE 2009, Rome, Italy, December 14-18, 2009)*, vol. 5929 of Lecture Notes in Computer Science, Springer, 2009.

[69] L. Lovász and M. D. Plummer, *Matching Theory*, North-Holland mathematics studies, North-Holland, Amsterdam, New York, 1986.

[70] A. Madry, *Navigating central path with electrical flows: From flows to matchings, and back*, in FOCS 2013 [43], pp. 253–262.

[71] M. Mahdian and Q. Yan, *Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs*, in Fortnow and Vadhan [45], pp. 597–606.

[72] V. H. Manshadi, S. O. Gharan, and A. Saberi, *Online stochastic matching: Online actions based on offline statistics*, in Randall [86], pp. 1285–1294.

[73] A. McGregor, *Finding graph matchings in data streams*, in Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques (Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and the 9th International Workshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005), C. Chekuri, K. Jansen, J. D. P. Rolim, and L. Trevisan, eds., vol. 3624 of Lecture Notes in Computer Science, Springer, 2005, pp. 170–181.

[74] A. Mehta, *Online matching and ad allocation*, Foundations and Trends in Theoretical Computer Science, 8 (2013), pp. 265–368.

[75] A. Mehta and D. Panigrahi, *Online matching with stochastic rewards*, in FOCS 2012 [42], pp. 728–737.

[76] A. Mehta, A. Saberi, U. V. Vazirani, and V. V. Vazirani, *Adwords and generalized on-line matching*, in Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2005, 23-25 October 2005, Pittsburgh, PA, USA, IEEE Computer Society, 2005, pp. 264–273.

[77] A. Mehta, B. Waggoner, and M. Zadimoghaddam, *Online stochastic matching with unequal probabilities*, in Indyk [61], pp. 1388–1404.

[78] S. Micali and V. V. Vazirani, *An $\mathcal{O}(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs*, in Proceedings of the 21st Annual Symposium on Foundations of Computer Science, Syracuse, NY, USA, 13-15 October 1980, IEEE Computer Society, 1980, pp. 17–27.

[79] M. Mucha and P. Sankowski, *Maximum matchings via gaussian elimination*, in Proceedings of the 45th Symposium on Foundations of Computer Science, FOCS 2004, 17-19 October 2004, Rome, Italy, IEEE Computer Society, 2004, pp. 248–255.

[80] O. Neiman and S. Solomon, *Simple deterministic algorithms for fully dynamic maximal matching*, in Proceedings of the Symposium on Theory of Computing Conference, STOC 2013, Palo Alto, CA, USA, June 1-4, 2013, D. Boneh, T. Roughgarden, and J. Feigenbaum, eds., ACM, 2013, pp. 745–754.

[81] K. Onak and R. Rubinfeld, *Maintaining a large matching and a small vertex cover*, in Schulman [89], pp. 457–464.

[82] M. Patrascu, *Towards polynomial lower bounds for dynamic problems*, in Schulman [89], pp. 603–610.

[83] J. Petersen, *Die theorie der regulären graphs*, Acta Mathematica, 15 (1891), pp. 193–220.

[84] M. Poloczek and M. Szegedy, *Randomized greedy algorithms for the maximum matching problem with new analysis*, in FOCS 2012 [42], pp. 708–717.

[85] M. O. Rabin and V. V. Vazirani, *Maximum matchings in general graphs through randomization*, J. Algorithms, 10 (1989), pp. 557–567.

[86] D. Randall, ed., *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, SIAM, 2011.

[87] P. Sanders, N. Sivadasan, and M. Skutella, *Online scheduling with bounded migration*, Math. Oper. Res., 34 (2009), pp. 481–498.

[88] P. Sankowski, *Faster dynamic matchings and vertex connectivity*, in Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007, N. Bansal, K. Pruhs, and C. Stein, eds., SIAM, 2007, pp. 118–126.

[89] L. J. Schulman, ed., *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, ACM, 2010.

[90] D. D. Sleator and R. E. Tarjan, *A data structure for dynamic trees*, Journal of Computer and System Sciences, 26 (1983), pp. 362–391.

[91] R. L. Thorndike, *The problem of classification of personnel*, Psychometrika, 15 (1950), pp. 215–235.

[92] V. V. Vazirani, *An improved definition of blossoms and a simpler proof of the MV matching algorithm*, CoRR, abs/1210.4594 (2012).

[93] J. Westbrook, *Load balancing for response time*, in Algorithms – ESA '95 (Proceedings of the 3rd Annual European Symposium, ESA '95, Corfu, Greece, September 25-27, 1995), P. G. Spirakis, ed., vol. 979 of Lecture Notes in Computer Science, Springer, 1995, pp. 355–368.