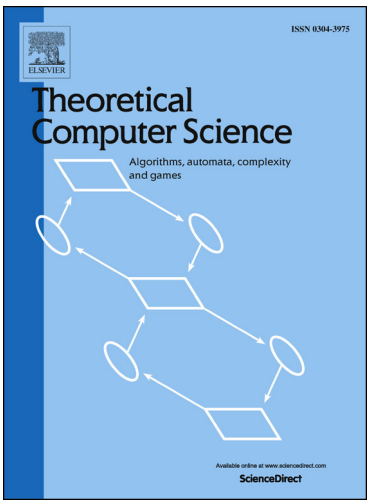
Accepted Manuscript



On extensions of the deterministic online model for bipartite matching and max-sat

N. Pena, A. Borodin

PII: S0304-3975(18)30632-7
DOI: https://doi.org/10.1016/j.tcs.2018.10.015
Reference: TCS 11773

To appear in: *Theoretical Computer Science*

Received date: 24 September 2017
Revised date: 21 May 2018
Accepted date: 15 October 2018

Please cite this article in press as: N. Pena, A. Borodin, On extensions of the deterministic online model for bipartite matching and max-sat, *Theoret. Comput. Sci.* (2018), https://doi.org/10.1016/j.tcs.2018.10.015

On extensions of the deterministic online model for bipartite matching and max-sat

N. Pena* A. Borodin†

October 17, 2018

Abstract

The surprising results of Karp, Vazirani and Vazirani [39] and (respectively) Buchbinder et al [18] are examples where rather simple randomization provides provably better approximations than the corresponding deterministic counterparts for online bipartite matching and (respectively) unconstrained non-monotone submodular. We show that seemingly strong extensions of the deterministic online computation model can at best match the performance of naive randomization. More specifically, for bipartite matching, we show that in the priority model (allowing very general ways to order the input stream), we cannot improve upon the trivial $\frac{1}{2}$ approximation achieved by any greedy maximal matching algorithm and likewise cannot improve upon this approximation by any $\frac{logn}{\log\log n}$ number of online algorithms running in parallel. The latter result yields an improved $\log\log n - \log\log\log n$ lower bound for the number of advice bits needed to beat the simple deterministic greedy algorithm. For max-sat, we adapt the recent de-randomization approach of Buchbinder and Feldman [16] applied to the Buchbinder et al [17] algorithm for max-sat to obtain a deterministic $\frac{3}{4}$ approximation algorithm using width $2n$ parallelism. In order to improve upon this approximation, we show that exponential width parallelism of online algorithms is necessary (in a model that is more general than what is needed for the width $2n$ algorithm). We relate our results to previous work concerning the priority Branching Tree (pBT) model of Alekhnovich et al [2].

Keywords: online algorithms; priority algorithms; advice; parallel online algorithms

1 Introduction

It is well known that in the domain of online algorithms it is often *provably* necessary to use randomization in order to achieve good approximation (competitive) ratios. It is interesting to ask when can the use of randomization be replaced by extending the online framework. In a more constructive sense, can we de-randomize certain online algorithms by considering more general one pass algorithms? This question has already been answered in a couple of senses. Böckenhauer et al [11] show that a substantial class of randomized online algorithms can be transformed (albeit *non-uniformly* and inefficiently) to an online algorithm with (small) advice. Buchbinder and Feldman [16] show how to uniformly and efficiently de-randomize the Buchbinder et al [17, 18] algorithm for the unconstrained non-monotone submodular function maximization (USM) problem. The

*Google, Waterloo Ontario, nicolaspe92@gmail.com.
†Dept of Computer Science,University of Toronto, bor@cs.toronto.edu.

resulting de-randomized algorithm in [16] can be viewed as a parallel algorithm in the form of a "tree of online algorithms". We formalize their algorithm as an online restriction of the Alekhnovich et al [2] pBT model.

In this paper we consider two classical optimization problems, namely maximum cardinality bipartite matching and the max-sat problem. As an offline problem, it is well known that graph matching and more specifically bipartite matching (in both the unweighted and weighted cases) can be solved optimally in polynomial time. Given its relation to the adwords problem for online advertising, bipartite matching has also been well-studied as a (one-sided) online problem. Max-sat is not usually thought of as an online problem but many of the combinatorial approximation algorithms for max-sat (e.g. Johnson's algorithm [37], Poloczek and Schnitger [49], and Buchbinder et al [18]) can be viewed as online or more generally one-pass myopic algorithms.

To study these types of problems from an online perspective, several precise models of computation have been defined. With respect to these models, we can begin to understand the power and limitations of deterministic and randomized algorithms that in some sense can be viewed as online or one-pass. Our paper is organized as follows. We conclude this section with an informal list of our main results. The necessary definitions will be provided in Section 2. Section 3 contains a review of the most relevant previous results. Section 4 considers online parallel width results for max-sat. In Section 5 we consider results for bipartite matching with respect to parallel width, the priority model [13], and the random order model (ROM). We conclude with a number of open problems in Section 6.

1.1 Our results

- For the max-sat problem, we show that the Buchbinder and Feldman [16] de-randomization method can be applied to obtain a deterministic parallel width $2n$ online $\frac{3}{4}$ approximation algorithm.
- We then show (in a model more general than what is needed for the above $\frac{3}{4}$ approximation), that exponential width is required to improve upon this ratio even for the exact max-2-sat problem for which Johnson's algorithm already achieves a $\frac{3}{4}$ approximation.
- For bipartite matching we show that constant width (or even width $\frac{\log n}{\log \log n}$) cannot asymptotically beat the trivial $\frac{1}{2}$ approximation achieved by any greedy maximal matching algorithm. This implies that more than $\log \log n - \log \log \log n$ advice bits are needed to asymptotically improve upon the $\frac{1}{2}$ approximation achieved by any greedy maximal matching algorithm.
- For bipartite matching, we also show that the ability to sort the input items as in the priority model cannot compensate for the absence of randomization.
- Finally, we make some observations about bipartite matching in the fully randomized priority model and the ROM model.

2 Preliminaries: basic definitions, input and algorithmic models

We briefly describe the problems of interest in the paper and then proceed to define the relevant algorithmic models and input models. present our results.

2.1 The Bipartite Matching and Max-Sat Problems

In the unweighted matching problem, the input is a graph $G = (V, E)$ and the objective is to find the largest subset of edges $S \subseteq E$ that are vertex-disjoint and such that $|S|$ is as large as possible. Bipartite matching is the special case where $V = A \cup B$ and $E \subseteq A \times B$.

In the (weighted) max-sat problem, the input is a propositional formula in CNF form. That is, there is a set of clauses $\mathcal{C} = \{C_1, \ldots, C_n\}$ where each clause is a set of literals and each literal is a propositional variable or its negation. In the unweighteed case, The objective is to find an assignment to the variables that maximizes the number of clauses in $\mathcal{C}$ that are satisfied. In the weighted case, each clause has an associated weight and the objective is to maximize the sum of weights of clauses that are satisfied. The max-sat problem has been generalized to the submodular max-sat problem where there is a normalized monotone submodular function $f : 2^{\mathcal{C}} \to \mathbb{R}$ and we wish to assign the variables to maximize $f(\mathcal{C}')$ where $\mathcal{C}' \subseteq \mathcal{C}$ is the subset of satisfied clauses.

When a specific problem needs to be solved, there are many possible input instances. An instance is just a specific input for the problem in hand. For example, in bipartite matching, an instance is a bipartite graph. In weighted max-sat, an instance is a set of clauses (including a name, weight, and the literals in each clause). An algorithm will have the goal of obtaining a good solution to the problem for every possible instance (i.e. we are only considering worst case complexity). To establish inapproximation results, we construct an adversarial instance or a family of bad instances for every algorithm (one instance or family of instances per algorithm). We mainly study deterministic algorithms and their limitations under certain models.

We measure the performance of online algorithms by the competitive (or approximation) ratio[1].

Definition 2.1. Let $\mathbb{A}$ be an algorithm for a problem. Let $\mathcal{I}$ be the set of all possible instances for the problem. For $I \in \mathcal{I}$, let $v(\mathbb{A}, I)$ be the value obtained by the algorithm $\mathbb{A}$ on instance I and let $v(I)$ be the optimal value for that instance. The approximation ratio of algorithm $\mathbb{A}$ is:

$$\inf_{I \in \mathcal{I}} \frac{v(\mathbb{A}, I)}{v(I)}$$

Let $\mathcal{I}_n$ be the set of all instances of size n for the problem. The asymptotic approximation ratio of $\mathbb{A}$ is:

$$\liminf_{n \in \mathbb{N}} \inf_{I \in \mathcal{I}_n} \frac{v(\mathbb{A}, I)}{v(I)}$$

The approximation ratio considers how the algorithm performs in every instance, while the asymptotic approximation ratio considers how the algorithm performs on large instances. Having a small instance where an algorithm performs poorly shows that the algorithm has a low approximation ratio, but says nothing about its asymptotic approximation ratio.

2.2 Algorithmic Models

We define the precise *one-pass* algorithmic models that we consider in this paper. For each model, the algorithm may receive some limited amount of information in advance. Other than this, an instance is composed of individual *data items*. We define the size of an instance as the number of

[1]The name competitive ratio is usually used when considering online problems while approximation ratio is used in other settings. We will just use approximation ratio in any model of computation. For the maximization problems we consider, the approximation ratio is typically considered as a fraction less than or equal to 1.

data items it is made of. Data items will be received in a certain order, and how this ordering is chosen depends on the algorithmic model. In addition, when a data item is received, the algorithm must make an irrevocable decision regarding this data item before the next data item is considered. The solution then consists of the decisions that have been made. We do not make any assumptions regarding time or space constraints for our algorithms. In fact, the limitations proven for these algorithmic models are information-theoretic: since the algorithm does not know the whole instance, there are multiple potential instances, and any decision it makes may be bad for some of these. Of course, how each data model is defined will depend on the specific problem (and even for one problem there may be multiple choices for the information contained in data items). Unless otherwise stated, for inapproximations we assume that the algorithm knows the input size (i.e. the number of variables in max-sat and number of online nodes in bipartite matching) of the problem, and for positive results we assume that it does not know the size.

2.2.1 Online Model

In the online model, the algorithm has no control whatsoever on the order in which the data items are received. That is, data items arrive in any order (in particular, they may arrive in the order decided by an adversary) and as each data item arrives, the algorithm has to make its decision. Thus, in this model an adversary chooses an ordering to prevent the algorithm from achieving a good approximation ratio. As long as it remains consistent with previous data items, the data item that an adversary presents to the algorithm may depend on previous data items and the choices that the algorithm has made concerning them. The following presents the structure of an online algorithm.

Online Algorithm
On an instance I, including an ordering of the data items $(d_1, \ldots, d_n)$:
$i := 1$
While there are unprocessed data items
    The algorithm receives d_i and makes an irrevocable decision for d_i
    (based on d_i and all previously seen data items and decisions).
    $i := i + 1$
EndWhile

In online bipartite matching, it is standard to consider that the algorithm knows in advance the names of the vertices from one of the sides of the graph, which we call the offline side. The other side, which we call the online side, is presented in parts. A data item consists of a single vertex from the online side along with all of its neighbours on the offline side. At each step, an algorithm can match the online vertex to any of its unmatched neighbours, or it can choose to reject the vertex (leave it unmatched). In either case, we say that it processes the vertex. All decisions made by the algorithm are irrevocable. We call this problem the one-sided online bipartite matching. We do not consider online matching for general graphs nor two-sided bipartite online matching where every node (or edge) is an input item.

Remark 2.2. As a specific example of an algoritm for the one-sided bipartite matching problem consider *the basic* deterministic (online) greedy algorithm where given any ordering of the offline nodes (say $v_1 < \ldots, < v_n$), each online node u_j gets matched to
$\text{argmin}_i\{v_i : v_i$ has not been matched to any $u_k, k < j\}$ if such an unmatched v_i exists and otherwise u_j will not be matched. As an example, consider the regular degree two 3×3 bipartite graph

where u_1 has neighbors, $\{v_1, v_3\}, u_2$ has neighbors $\{v_2, v_3\}$, and u_3 has neighbours $\{v_1, v_2\}$. The basic greedy algorithm will match u_1 to v_1, u_2 to v_2 forcing u_3 to be unmatched even though there is a perfect matching.

In the max-sat problem, a data item consists of a single propositional variable along with some information about the clauses (called relevant clauses) where this variable appears. We consider four models, in increasing order of the amount of knowledge received:

Model 0: The data item is a list with the names and weights of the relevant clauses where the variable appears positively and a list of names and weights of the clauses where the variable appears negatively.

Model 1: Model 0, and in addition, the length of each relevant clause is also included.

Model 2: Model 1, and, in addition, for each relevant clause we also include the names of the other variables that occur in this clause. The data item does not include the signs of these variables.

Model 3: Model 2, plus for each relevant clause we also include the signs of other variables in the clause. That is, in this model, the data item contains all the information about the clauses in which the variable occurs.

For submodular max-sat, in addition to a description of the model for variables and their clauses, we need to state how the submodular function is presented to the algorithm. Since the submodular function has a domain exponential in the size of the ground set (in this case the number of clauses), it is usually assumed that the algorithm does not receive the whole description of this function from the start. Instead, there is an oracle which will answer queries that the algorithm makes. A common oracle model is the value oracle, where the algorithm may query $f(S)$ for any subset of clauses S. In our restricted models of computation, this is further restricted so that the set S can include clauses containing at least one variable seen so far, including the current element for which the algorithm is making a decision. However, for some algorithms this model is too restrictive. More specifically, in order to abstract the Buchbinder et al [18] "greedy algorithm" for non-monotone submodular maximization, Huang and Borodin [34] introduce a "double-sided myopic model" in which the value oracle may also query the complement function $\overline{f}$ defined by $\overline{f}(S) = f(C \setminus S)$ (where C is the ground set).

2.2.2 Width models

We consider a framework which provides a natural way to allow more powerful algorithms, while maintaining in some sense an online setting. The idea is that now, instead of maintaining a single solution, the algorithm keeps multiple possible solutions for the instance. When the whole instance has been processed, the algorithm returns the best among the set of possible solutions that it currently has. The computational issue is to study the tradeoff between the number of possible solutions the algorithm maintains at any point in time and the quality of the best solution that will be the final output. The computational model can be viewed as a levelled tree where each level corresponds to an iteration of the algorith. A node in the tree corresponds to a possible solution and an edge means that one solution led to the other one after the algorithm saw the next data item. At the beginning, there is a single empty solution: the algorithm has made no choices yet. This will be the root of the tree and it will be at level 0. Every time a possible solution is split into multiple solutions, the tree branches out. Every time a data item is processed, the level increases by 1. The algorithm may decide to discard some solution. This corresponds to a node with no children and we say that the algorithm cuts the node. In all these models, we may want to limit

the maximum width k in each level. Since each level represents a point in time, this means the algorithm can never maintain more than k possible solutions.

There are three different models to consider here. In the *max-of-k model*, the algorithm branches out at the very beginning and does not branch or cut later on. In particular, this model is the same as having k different algorithms and taking the maximum over all of them in the end. In the *width-k model*, the algorithm can branch out at any time. However, the algorithm cannot cut any possible solution: every node in the tree that does not correspond to a complete solution (obtained after viewing all data items) must have at least one child. Finally, in the *width-cut-k model*, the algorithm can branch out and disregard a possible solution at any time (so that, at later levels, this possible solution will not contribute to the width count). The width-cut-k model is the online restriction of the fixed order Alekhnovich [2] pBT model where the algorithm can initially choose an ordering of the input items rather than the order being determined adversarially as in the standard (i.e. width 1) online model.

Although we shall not do so, the width models can also be extended to the priority and ROM settings (see sections 2.2.4 and 2.2.5). For example, we can consider width in the priority model as follows: the algorithm can choose an ordering of the data items as in the fixed priority model (in particular, the arrival order has to be the same for all partial solutions), and once an input item arrives the solutions are updated, branch, or cut as desired. This then is the fixed order pBT model of Alekhnovich et al [2]. It is also possible to allow each branch of the pBT to adaptively choose the ordering of the remaining items and this then is the adaptive pBT model.

2.2.3 Relation to advice and semi-streaming models

Most inapproximation results for the online and related models we consider allow the algorithm to know n, the size of the input. One would like to allow these algorithms to also know other easily computed information about the input (e.g. maximum or minimum degree of a node, etc.) In keeping with the information theoretic nature of inapproximation results, one way to state this is to allow these algorithms to use any small (e.g. $O(\log n)$) bits of advice and not require that the advice be efficiently computed. The *online with advice model* [15, 11, 24] that is related to our width models allows the algorithm to access a (say) binary advice string initially given to it by an oracle that knows the whole input and has unlimited computational power. Clearly, the advice string could encode the optimal strategy for the algorithm on the input, so the idea is to understand how the performance of the algorithm changes depending on how many bits of advice it uses. A non-uniform algorithm is a set of algorithms, one for each value of n. In particular, a non-uniform algorithm knows n, the size of the input. The following simple observation shows that the non-uniform advice and max-of-k models are equivalent.

Lemma 1. *Suppose that an online algorithm knows n, the size of the input. Then there is an algorithm using $b(n)$ advice achieving an approximation ratio of c if and only if there is a max-of-$2^{b(n)}$ algorithm with approximation ratio c.*

Proof. Let $\mathbb{A}$ be an online advice algorithm using advice of size t and achieving a c approximation ratio. The following max-of-2^t algorithm achieves this ratio: try all possible advice strings of length t, and take the best option among these. If $\mathbb{M}$ is a max-of-2^t algorithm with approximation ratio c, then there is a t bit advice algorithm that, for each input, encodes the best choice among the 2^t that will achieve this approximation ratio. □

The online advice model and therefore the small width online model also has a weak relation with the graph semi-streaming graph, a model suggested by Muthukrishhan [46] and studied further in Feigenbaum et al [26]. In that model, *edges* arrive online for a graph optimization problem (e.g. matching). More directly related to our models, Goel et al [28] consider the model where vertices arrive online. In either case, letting n be the number of vertices, the algorithm is constrained to use $\tilde{O}(n) = n \log^{O(1)} n$ space which can be substantially less space than the number of edges. The online advice and semi-streaming models are not directly comparable since on the one hand semi-streaming algorithms are not forced to make irrevocable decisions, while online algorithms are not space constrained. In order to relate the online model to the semi-streaming model, we need to restrict the online model to those algorithms in which the computation satisfies the $\tilde{O}(n)$ semi-streaming space bound. In particular, the algorithm cannot store all the information contained in the data items that have been considered in previous iterations.

2.2.4 Priority model

The difference between the priority and online models is that in the priority model the algorithm has some power over the order in which data items arrive. Every time a new data item is about to arrive, an ordering on the universe of potential data items is used to decide which one arrives next. This ordering does not impose any restrictions on how this order is produced except that it cannot depend on data items from the current instance that it has not yet seen. In particular, the ordering need not be computable. In the *fixed priority model*, the algorithm only provides one ordering at the beginning. For a given instance, the data items are shown to the algorithm according to this ordering (and as before, when one arrives, the algorithm must make a decision regarding the data item).

Fixed Priority Algorithm

1: The algorithm specifies an ordering $\pi : U \to \mathbb{R}$, where U is the universe of all possible data items.
2: On instance I, the data items are ordered $d_1, \ldots, d_n$ so that $\pi(d_1) \leq \pi(d_2) \leq \ldots \leq \pi(d_n)$.
3: $i := 1$
4: **While** there are unprocessed data items
5: The algorithm receives d_i and makes a decision for d_i
(based on d_i and all previously seen data items and decisions).
6: $i := i + 1$
7: **EndWhile**

In the *adaptive priority model*, the algorithm provides a new ordering every time a data item is about to arrive. Thus, in the fixed order model, the priority of each item is a (say) real valued function of the input item and in the adaptive order model, the priority function can also depend on all previous items and decisions. In the matching problem, an algorithm could for example choose an ordering π so that data items corresponding to low degree vertices are preferred. Or it could choose π to prefer data items corresponding to vertices that are neighbours of some specific vertex. Unless otherwise stated, when we say priority we mean adaptive priority. The following shows the template for adaptive priority algorithms:

Adaptive Priority Algorithm

On instance I, initialize D as the set of data items corresponding to I and U as the universe of all data items. Let $n := |D|$.
$i := 1$
While there are unprocessed data items
  The algorithm specifies an ordering $\pi : U \to \mathbb{R}$.
  Let $d_i := \text{argmin}_{d \in D}(\pi(d))$.
  The algorithm receives d_i and makes a decision about it.
  Update: $D \leftarrow D \setminus \{d_i\}$, $U \leftarrow \{\text{data items consistent with } d_1, \ldots, d_i\}$.
  $i := i + 1$
EndWhile

Usually, the adversarial argument in this model is as follows: the adversary begins by choosing a subset S from the universe of all data items. This will be the set of potential data items for the problem instance. Now, for every $s_1 \in S$, the algorithm could choose this data item as the first one by using an appropriate ordering π. After the algorithm chooses s_1 and makes its decision for this data item, the adversary further shrinks S, thus obtaining a smaller subset of potential data items. The algorithm then proceeds by choosing a second data item $s_2 \in S$ and makes a decision concerning it. After this the adversary further shrinks S, and this goes on until S becomes empty.

In some adversarial instances, some data items may be indistinguishable to the algorithm. If d and d' are indistinguishable to the algorithm, the algorithm may produce an ordering π to try to receive d, but the adversary can force d' to be received instead. For instance, in the matching problem, with regard to deterministic priority algorithms, initially data items corresponding to vertices with the same degree will be indistinguishable. If the algorithm tries to produce an ordering π to get the data item of a specific vertex, the adversary can rename vertices so that the algorithm receives the data item of some other vertex with the same degree. However, after the first vertex has been seen, the algorithm now does have some limited information about the names of other nodes and can possibly exploit this knowledge. In contrast, we note that this degree-based argument can fail for randomized priority algorithms as we will see in Section 5.2.2.

In the most common model for general graph matching, a data item consists of a vertex name along with its neighbours. Here, every vertex has an associated data item. This contrasts with the common data item for online bipartite matching, where only the online side vertices have associated data items. We call the former model (restricted to instances that are bipartite) two-sided bipartite matching, and the latter one-sided. We shall restrict attention to the one-sided problem.

2.2.5 The random order model

In the random order model, usually abbreviated by ROM, neither the algorithm nor the adversary chooses the order in which the data items are presented. Instead, given an input set chosen by an adversary, a permutation of the data items is chosen uniformly at random and this permutation dictates the order in which data items are presented. Once the random input permutation has been instantiated, the model is the same as the one-sided online model.

Definition 2.3. Let $\mathbb{A}$ be an online algorithm for a problem whose set of all possible instances is $\mathcal{I}$. For an instance I of size n and a permutation $\sigma \in S_n$, let $I(\sigma)$ be I with data items presented in the order dictated by σ. Let $v(\mathbb{A}, I(\sigma))$ be the value achieved by the algorithm on $I(\sigma)$, and let

$v(I)$ be the optimal value for I. The approximation ratio of $\mathbb{A}$ in ROM is:

$$\inf_{I \in \mathcal{I}} \frac{\mathbb{E}_{\sigma \in S_n}[v(\mathbb{A}, I(\sigma))]}{v(I)}$$

Let $\mathcal{I}_n$ be the set of instances of size n. Then the asymptotic approximation ratio of $\mathbb{A}$ in ROM is:

$$\liminf_{n \in \mathbb{N}} \inf_{I \in \mathcal{I}_n} \frac{\mathbb{E}_{\sigma \in S_n}[v(\mathbb{A}, I(\sigma))]}{v(I)}$$

3 Related work

The analysis of online algorithms in terms of the competitive ratio was explicitly begun by Sleator and Tarjan [53] although there were some previous papers that implicitly were doing competitive analysis (e.g. Yao [56]). The max-of-k online width model was introduced in Halldórsson et al [31] and Iwama and Taketomi [35] where they considered the the maximum independent set and knapsack problems. Buchbinder and Feldman showed a deterministic algorithm with approximation ratio 1/2 for unconstrained submodular maximization which fits the online width model (but not the max-of-k model) [16]. Their initial approach involved solving an LP at each step. They showed how to simplify the algorithm so that it does not require an LP solver, and the width used is linear.

Hopcroft and Karp [33] showed that unweighted bipartite matching can be optimally solved offline in time $O(m\sqrt{n})$. For sparse graphs, the first improvement in 40 years is the $\tilde{O}(m^{\frac{10}{7}})$ time algorithm due to Madry [41]. With regard to the online setting, the seminal paper of Karp, Vazirani and Vazirani [39] established a number of surprising results for (one-sided) online bipartite matching. After observing that no online deterministic algorithm can do better than a $\frac{1}{2}$ approximation, they studied randomized algorithms. In particular they showed that the natural randomized algorithm RANDOM (that matches an online vertex uniformly at random to an available offline vertex) only achieved an asymptotic approximation ratio of $\frac{1}{2}$; that is, the same approximation as any greedy maximal matching algorithm. They then showed that their randomized RANKING algorithm achieved [2] ratio $1 - \frac{1}{e} \approx .632$. RANKING initially chooses a permutation of the offline vertices and uses that permutation to determine how to match an online vertex upon arrival. By deterministically fixing any permutation of the offline vertices, the Ranking algorithm can be interpreted as a deterministic algorithm in the ROM model. While Ranking is optimal as an online randomized algorithm, it is not known if its interpretation as a deterministic ROM algorithm is optimal for all online deterministic algorithms. Goel and Mehta [29] show that no deterministic ROM algorithm can achieve an approximation better than $\frac{3}{4}$. The KVV algorithm can also be implemented as a $O(n \log n)$ space randomized semi-streaming algorithm whereas Goel el al [28] show that there is a deterministic semi-streaming algorithm using only $O(n)$ space provably establishing the power of the semi-streaming model. In the ROM model, the randomized Ranking algorithm achieves an approximation ratio of at least $0.696 > 1 - 1/e$ [38, 42] and at most 0.727 [38]. Following the KVV paper there have been a number of extensions of online bipartite matching with more direct application to online advertising (see, for example, [44, 29, 1, 40]), and has also been studied in various stochastic models where the input graph is generated by sampling i.i.d from a known or unknown distribution of online vertices (see [27, 7, 43, 36, 38]). Manshadi et al [43] showed that

[2] It was later discovered [29] (and independently by Krohn and Varadarajan) that there was an error in the KVV analysis. A correct proof was provided in [29] and subsequently alternative proofs [10, 22] have been provided.

no randomized ROM algorithm can achieve an asymptotic approximation ratio better than 0.823 by establishing that inapproximation for the stochastic unknown i.i.d model.

In the online with advice model for bipartite matching, Dürr et al [23] apply the Böckenhauer et al de-randomization idea to show that for every $\epsilon > 0$, there is an (*inefficient*) $O(\log n)$ advice algorithm achieving ratio $(1-\epsilon)(1-\frac{1}{e})$. This is complemented by Mikkleson's [45] recent result showing that no online (even randomized) algorithm using sublinear $o(n)$ advice can asymptotically improve upon the $1-\frac{1}{e}$ ratio achieved by KVV. Furthermore, Dürr et al show that $O(\frac{1}{\epsilon^5}n)$ advice is sufficient, and $\Omega(\log(\frac{1}{\epsilon})n$ advice is necessary to achieve a $(1-\epsilon)$ ratio. They show that for a natural but restricted class of algorithms, $\Omega(\log\log\log n)$ advice bits are needed for deterministic algorithms to obtain an approximation ratio asymptotically better than 1/2. Finally, their Category-Advice algorithm is a deterministic two pass online (i.e. adversarial order) $\frac{3}{5}$ approximation algorithm where the first pass is used to give priority in the second pass to the offline vertices that were unmatched in the first pass. That is, the first pass is efficiently constructing an n bit advice string for the second pass.

The priority setting was introduced by Borodin et al [13]. It has been studied for problems such as makespan scheduling [3], and also in several graph optimization problems [12, 21]. In terms of the maximum matching problem, most results are about general graph matching. Here, a data item consists of a vertex (the vertex that needs to be matched) along with a list of its neighbours. Aronson et al [4] showed that the algorithm which at each step chooses a random vertex and then a random neighbour (to pick an edge to add to the matching) achieves an approximation ratio of $1/2+c$ for some $c > 0$. Besser and Poloczek [8] showed that MinGreedy, the algorithm that at each step picks an edge with a vertex of minimum degree, will not get an asymptotic approximation ratio better than 1/2. In contrast they show that for d-regular graphs the MinGreedy approximation ratio improves to $\frac{d-1}{2d-3}$. They also showed that no deterministic greedy (adaptive) priority algorithm can beat this ratio for graphs of maximum degree d (which implies that these algorithms cannot get an approximation ratio greater than 1/2), and they showed no deterministic priority algorithm (whether or not greedy) can get an approximation ratio greater than 2/3 (5/6 for "degree based" randomized algorithms). We note that the positive approximation bounds immediately apply to bipartite graphs in the two-sided model where input nodes come from both sides but do not necessarily carry over to our one-sided online model. Similarly, Besser and Poloczek show that their inapproximation for MinGreedy can be made to apply to bipartite graphs in the two-sided model but again this does not necessarily imply the same inapproximation in the one-sided model.

Håstad [32] showed that it is NP-hard to achieve an approximation ratio of c for the maximum satisfiability problem for any constant $c > 7/8$, and the best known efficient algorithm has an approximation ratio of 0.797 and a conjectured approximation ratio of 0.843 [5]. The greedy algorithm that at each step assigns a variable to satisfy the set of clauses with larger weight is an online algorithm achieving an approximation ratio of 1/2. Azar et al [6] observed that this is optimal for deterministic algorithms with input model 0. They showed a randomized greedy algorithm that achieves an approximation ratio of 2/3 for online submodular max-sat, and they showed that this is optimal for input model 0. In this algorithm, when a variable arrives, the variable is set to true with probability $\frac{w_T}{w_T+w_F}$ and set to false otherwise, where w_T is the weight of clauses satisfied if assigned to true and w_F is the weight of clauses satisfied if assigned to false. For submodular max-sat, the weight is replaced by the marginal gain.

Johnson's algorithm [37] is a deterministic greedy algorithm that bases its decisions on the "measure of clauses" satisfied instead of the weights of these clauses. Yannakakis [55] showed that

Johnson's algorithm is the de-randomization (by the method of conditional expectations) of the naive randomized algorithm and also showed that no deterministic algorithm can achieve a better approximation ratio even in input model 3. Chen et al [19] showed that Johnson's algorithm achieves this 2/3 approximation ratio. The analysis was later simplified by Engebretsen [25]. Johnson's algorithm can be implemented in input model 1. Costello et al [20] showed that Johnson's algorithm achieves an approximation ratio of $2/3 + c$ for some $c > 0$ in ROM. Poloczek and Schnitger gave an online randomized algorithm in input model 1 achieving an approximation ratio of 3/4 [49]. They showed that Johnson's algorithm in ROM gets an approximation ratio of at most $2-\sqrt{15} < 3/4$, and they showed that the online randomized version of Johnson's algorithm (which assigns probabilities according to measures, as in the randomized greedy algorithm) achieves an approximation ratio of at most $17/23 < 3/4$.

Van Zuylen gave a simpler online randomized algorithm [54] with approximation ratio 3/4. Buchbinder et al [17, 18] gave a randomized algorithm for unconstrained submodular maximization with approximation ratio 1/2 and additionally a related randomized algorithm for submodular max-sat achieving an approximation ratio of 3/4. Poloczek [48] showed that no deterministic adaptive priority for max-sat can achieve an approximation ratio greater than $\frac{\sqrt{33}+3}{12} < 3/4$ in input model 2. He also showed that, under this input model, no randomized online algorithm can get an approximation ratio better than 3/4, so several algorithms achieving an approximation ratio of 3/4 that fit the framework are optimal (up to lower order terms). Yung [57] showed that no deterministic priority algorithm for max-sat in input model 3 can achieve an approximation ratio better than 5/6.

By extending the online framework, Poloczek et al [50] proposed a deterministic max-sat algorithm achieving a 3/4 approximation ratio that makes two passes over the input: in one pass, the algorithm computes probabilities for each variable, and in the second pass, it uses these probabilities and the method of conditional expectations to assign variables. Poloczek and Williamson [51] provide extensive simlulations showing that the two pass algorithm by itself and as a preprocessor for more complex algorithmsis is useful in practice in providing solutions that often are close to optimal. Dürr et al provide a deterministic two pass algorithm for bipartite matching that achieves approximation ratio 3/5. This is extended in Borodin et al [14] to multiple passes with approximation ratio limiting to $\approx .618$.

A model for priority width-and-cut was presented and studied by Alekhnovich et al [2]. In particular, they showed that deterministic fixed priority algorithms require exponential width to achieve an approximation ratio greater than 21/22 for max-sat in input model 3. We note that Alekhnovich et al did not consider the bipartite matching problem in the context of priority based models.

4 Max-sat width results

We first show that the Buchbinder and Feldman [16] de-randomization approach can be utillized to obtain a $\frac{3}{4}$ approximation by a parallel online algorithm of width $2n$. Then we show with respect to what we are calling input model 2, that exponential width is needed to improve upon this approximation.

4.1 Derandomizing the Buchbinder et al submodular max-sat algorithm

Buchbinder et al [18] presented a randomized algorithm for submodular max-sat with an approximation ratio of 3/4. They define a *loose assignment* of a set of variables V as a set $A \subseteq V \times \{0,1\}$. Any variable can be assigned one truth value (0 or 1), none, or both. A clause is satisfied by A if A contains at least one of the literals in the clause. For instance, $V \times \{0,1\}$ will satisfy any clause and $\emptyset$ will satisfy no clause. Let F be the normalized monotone submodular function on sets of clauses (this is part of the input to the problem) and let $g : \mathcal{P}(V \times \{0,1\}) \to \mathbb{R}$ be the function defined by $g(A) = F(C)$ where C is the set of clauses satisfied by the loose assignment A. It is easy to check that g is also a monotone submodular function.

The algorithm keeps track of a pair of loose assignments (X, Y), which change every time a new variable is processed. Let (X_i, Y_i) be the values right after the ith variable v_i is processed. Initially the algorithm begins by setting $X_0 = \emptyset$ and $Y_0 = V \times \{0,1\}$. When processing propositional variable v_i, the algorithm will decide whether to set $v_i = 1$ (i.e. true) or set $v_i = 0$ (i.e. false). Let b_i be the truth assignment (as determined below) for v_i. Then X_i will be $X_{i-1} \cup \{(v_i, b_i)\}$ while Y_i will be $Y_{i-1} \setminus \{(v_i, Y_{i-1})\} \cup \{(v_i, b_i)\}$. We say in this case that v_i is assigned b_i. Thus X_i and Y_i have the same unique assignment for the first i variables, X_i only contains assignments for the first i variables, and Y_i contains all possible assignments for the variables after v_i. If there are n variables, $X_n = Y_n$ is a proper assignment, and this is the output of the algorithm.

When processing v_i, the algorithm makes a random decision based on the marginal gains of assigning v_i to 0 and of assigning v_i to 1. The value $g(X_{i-1} \cup \{(v_i, 0)\}) - g(X_{i-1})$ is how much is gained by assigning v_i to 0, while $g(Y_{i-1}) - g(Y_{i-1} \setminus \{(v_i, 1)\})$ is how much is surely lost by assigning v_i to 0. Thus, the quantity $f_i := g(X_{i-1} \cup \{(v_i, 0)\}) - g(X_{i-1}) + g(Y_{i-1} \setminus \{(v_i, 1)\}) - g(Y_{i-1})$ is a value measuring how favourable it is to assign v_i to 0. Similarly, $t_i := g(X_{i-1} \cup \{(v_i, 1)\}) - g(X_{i-1}) + g(Y_{i-1} \setminus \{(v_i, 0)\}) - g(Y_{i-1})$ measures how favorable the assignment of v_i to 1 is. In the algorithm presented in [18], v_i is assigned to 0 with probability $\frac{f_i}{f_i + t_i}$ and to 1 with probability $\frac{t_i}{f_i + t_i}$ (with some care to avoid negative probabilities).

We now de-randomize this algorithm at the cost of having linear width. The de-randomization idea follows along the same lines as that of Buchbinder and Feldman [16] for a deterministic algorithm for unconstrained submodular maximization with a 1/2 approximation ratio. The authors present the novel idea of keeping a distribution of polynomial support over the states of the randomized algorithm. Normally, a randomized algorithm has a distribution of exponential support, so the idea is to carefully choose the states that are kept with nonzero probability. Elements of the domain (or in our case, variables) are processed one at a time, and at each iteration a linear program is used to determine the changes to the distribution. They then argue that they can get rid of the LP's to obtain an efficient algorithm, since solving them reduces to a fractional knapsack problem. The same LP format used for unconstrained submodular maximization works for submodular maxsat (the only change in the linear program in our algorithm below are the coefficients), so the idea in [16] to get rid of the LP solving also works for our algorithm.

Theorem 1. *There is a linear-width double-sided online algorithm for submodular max-sat achieving an approximation ratio of* 3/4*. The algorithm uses input model 1 of max- sat.*

Proof. First, we note that an oracle for F suffices for constructing an oracle for g. The algorithm keeps track of a distribution over pairs (X, Y) of loose assignments of variables. A double-sided algorithm is needed to obtain the values of the $g(Y)$'s. The idea is to process the variables online, at each step changing the distribution. The pairs (X, Y) satisfy the same properties as in the

Buchbinder et al algorithm. Thus, X corresponds to the assignments made in the partial solution so far, while Y corresponds to this plus the set of potential assignments that the partial solution could still make. When all variables are processed, the support will contain proper assignments of variables, and the algorithm takes the best one. The distribution is constructed by using an LP (without an objective function) to ensure some inequalities hold while not increasing the support by too much. We use the notation $(p, X, Y) \in D$ to say the distribution D assigns (X, Y) probability p. Also, if $(X, Y) \in supp(D_{i-1})$, we use the notation $Pr_{D_{i-1}}[X, Y]$ to denote the probability of the pair (X, Y) under distribution D_{i-1}. The variables are labelled $V = \{v_1, \ldots, v_n\}$ in the online order. See Algorithm 1.

Algorithm 1 Submodular Max Sat

1: Let $D_0 = \{(1, \emptyset, V \times \{0, 1\})\}$
2: **for** $i = 1$ to n **do**
3: $\forall (X, Y) \in supp(D_{i-1})$ let

$$\begin{aligned} f_i(X,Y) &= g(X \cup \{(v_i, 0)\}) - g(X) + g(Y \setminus \{(v_i, 1)\}) - g(Y) \\ t_i(X,Y) &= g(X \cup \{(v_i, 1)\}) - g(X) + g(Y \setminus \{(v_i, 0)\}) - g(Y) \end{aligned}$$

4: Obtain an extreme point solution for:

$$\begin{aligned} \mathrm{E}_{D_{i-1}}[z(X,Y)f_i(X,Y) + w(X,Y)t_i(X,Y)] &\geq 2\mathrm{E}_{D_{i-1}}[z(X,Y)t_i(X,Y)] && (1) \\ \mathrm{E}_{D_{i-1}}[z(X,Y)f_i(X,Y) + w(X,Y)t_i(X,Y)] &\geq 2\mathrm{E}_{D_{i-1}}[w(X,Y)f_i(X,Y)] && (2) \\ z(X,Y) + w(X,Y) &= 1 \ \forall (X,Y) \in supp(D_{i-1}) && (3) \\ z(X,Y), w(X,Y) &\geq 0 \ \forall (X,Y) \in supp(D_{i-1}) && (4) \end{aligned}$$

5: Construct a new distribution::

$$\begin{aligned} D_i = \quad & \{(z(X,Y)Pr_{D_{i-1}}[X,Y], X \cup \{(v_i,0)\}, Y \setminus \{(v_i,1)\}) : (X,Y) \in supp(D_{i-1})\} \\ & \cup \{(w(X,Y)Pr_{D_{i-1}}[X,Y], X \cup \{(v_i,1)\}, Y \setminus \{(v_i,0)\}) : (X,Y) \in supp(D_{i-1})\} \end{aligned}$$

6: Delete from D_i any pair of loose assignments with zero probability.
7: **end for**
8: **return** $\mathrm{argmax}_{(X,Y) \in supp_{D_n}}\{g(X)\}$

As before, $f_i(X, Y)$ is used to determine how profitable it is to assign v_i to 0 in this pair, and similarly $t_i(X, Y)$ measures how profitable it is to assign v_i to 1 in this pair. In normal max-sat, $f_i(X, Y)$ will be the weights of clauses satisfied by assigning v_i to 0 minus the weights of clauses that become unsatisfied by this assignment (a clause becomes unsatisfied when it hasn't been satisfied and all of its variables have been assigned), and $t_i(X, Y)$ is the analogue for the assignment to 1.

Each $(X, Y) \in supp(D_{i-1})$ will potentially be split into two in D_i: $(X \cup \{(v_i, 0)\}, Y \setminus \{(v_i, 1)\})$, corresponding to assigning v_i to 0 in pair (X, Y), and $(X \cup \{(v_i, 1)\}, Y \setminus \{(v_i, 0)\})$, corresponding to assigning v_i to 1 in pair (X, Y). $z(X, Y)$ is the probability of assigning v_i to 0, given (X, Y). Similarly, $w(X, Y)$ is the probability of assigning v_i to 1, given (X, Y). Since at each step D_i could grow twice the size, the LP is used to determine values for $z(X, Y)$, $w(X, Y)$ for all $(X, Y) \in$

$supp(D_{i-1})$ such that the resulting distribution still satisfies the properties used to achieve a good assignment in expectation while forcing many of the variables to be 0. In Step 6, the pairs with zero probability are trimmed from D_i to keep the distribution size small.

First, note that the distributions are well defined by induction and by inequalities 3 and 4 of the LP. Also, D_n contains well-defined assignments (instead of loose assignments), so the algorithm returns a valid assignment. Let $|D_i|$ be the size of the support of D_i. Excluding inequalities 4 stating the non-negativity of variables, there are $|D_{i-1}| + 2$ inequalities in the LP for step i, so an extreme point solution contains at most that many nonzero variables: $|D_i| \leq |D_{i-1}| + 2$. Therefore, $|D_n| \leq 2n + 1$ and the algorithm does have linear width.

Now, let us see that $t_i(X, Y) + f_i(X, Y) \geq 0$ for any $(X, Y) \in supp(D_{i-1})$. It can be proved by induction that for all $1 \leq i \leq n$ and all $(X, Y) \in supp(D_{i-1})$, $X \subseteq Y \setminus \{(v_i, 1), (v_i, 0)\}$. Then by submodularity,

$$g(X \cup \{(v_i, 0)\}) - g(X) \geq g(Y) - g(Y \setminus \{(v_i, 0)\})$$
$$g(X \cup \{(v_i, 1)\}) - g(X) \geq g(Y) - g(Y \setminus \{(v_i, 1)\})$$

Adding both inequalities, moving all terms to the left side and rearranging, we obtain that $t_i(X, Y) + f_i(X, Y) \geq 0$.

We now prove that for every i the LP formed is feasible, which is assumed by the algorithm in order to find an extreme point solution. We give an explicit feasible solution:

$$z_i(X, Y) = \frac{\max\{0, f_i(X, Y)\}}{\max\{0, f_i(X, Y)\} + \max\{0, t_i(X, Y)\}} \quad w_i(X, Y) = 1 - z(X, Y)$$

In case $t_i(X, Y) = f_i(X, Y) = 0$, we take $z(X, Y) = 1$ and $w(X, Y) = 0$. By definition, equalities 3 and inequalities 4 hold. When $t_i(X, Y) = f_i(X, Y) = 0$, the corresponding variables will not contribute to either side of inequalities 1 and 2. Assume either $t_i(X, Y) \neq 0$ or $f_i(X, Y) \neq 0$ and we want to show inequality 1 (since the other one will be analogous). Let $D := \max\{0, f_i(X, Y)\} + \max\{0, t_i(X, Y)\}$. Then we want to show:

$$\mathrm{E}\left[\frac{\max\{0, f_i(X, Y)\}}{D} f_i(X, Y) + \frac{\max\{0, t_i(X, Y)\}}{D} t_i(X, Y)\right] \geq 2\mathrm{E}\left[\frac{\max\{0, f_i(X, Y)\}}{D} t_i(X, Y)\right]$$

Because $D > 0$ this is equivalent to

$$\mathrm{E}\left[\max\{0, f_i(X, Y)\} f_i(X, Y) + \max\{0, t_i(X, Y)\} t_i(X, Y)\right] \geq 2\mathrm{E}\left[\max\{0, f_i(X, Y)\} t_i(X, Y)\right]$$

where the expectation is over D_{i-1}.

For $(X, Y) \in supp(D_{i-1})$ for which $f_i(X, Y) < 0$, we have $t_i(X, Y) > 0$ and the inequality becomes $t_i(X, Y)^2 \geq 0$, which clearly holds. Similarly, when $t_i(X, Y) < 0$ we must have $f_i(X, Y) > 0$ and the inequality becomes $f_i(X, Y)^2 \geq 2 f_i(X, Y) t_i(X, Y)$ which is true because the right hand side is negative. Finally, when $f_i(X, Y) \geq 0$ and $t_i(X, Y) \geq 0$, the inequality becomes $f_i(X, Y)^2 + t_i(X, Y)^2 \geq 2 f_i(X, Y) t_i(X, Y)$, which is true because $(a - b)^2 = a^2 + b^2 - 2ab \geq 0$.

Let OPT be an optimal assignment. For any $1 \leq i \leq n$ and $(X, Y) \in supp(D_i)$, let $OPT_i(X, Y) := (OPT \cup X) \cap Y$: it is an assignment that coincides with X and Y in the first i variables and coincides with OPT in the rest. Following the ideas in Buchbinder and Feldman [16], we prove the following main lemma:

Lemma 2. *For $1 \leq i \leq n$:*

$$\mathrm{E}_{D_{i-1}}[g(OPT_{i-1}(X,Y))] - \mathrm{E}_{D_i}[g(OPT_i(X,Y))] \leq \frac{1}{2}\left(E_{D_i}[g(X)+g(Y)] - E_{D_{i-1}}[g(X)+g(Y)]\right)$$

Proof. First suppose that in OPT, v_i is assigned 0. In this case:

$$\begin{aligned}
\mathrm{E}_{D_{i-1}}[g(OPT_{i-1}(X,Y))] \;-\; & \mathrm{E}_{D_i}[g(OPT_i(X,Y))] = \sum_{(X,Y)\in supp(D_{i-1})} [Pr_{D_{i-1}}[X,Y]g(OPT_{i-1}(X,Y)) \\
& - \; z_i(X,Y)Pr_{D_{i-1}}[X,Y]g\left(OPT_i(X\cup\{(v_i,0)\}, Y\setminus\{(v_i,1)\})\right) \\
& - \; w_i(X,Y)Pr_{D_{i-1}}[X,Y]g\left(OPT_i(X\cup\{(v_i,1)\}, Y\setminus\{(v_i,0)\})\right)] \\
= & \sum_{(X,Y)\in supp(D_{i-1})} w_i(X,Y)Pr_{D_{i-1}}[X,Y][g(OPT_{i-1}(X,Y)) \\
& - \; g\left(OPT_i(X\cup\{(v_i,1)\}, Y\setminus\{(v_i,0)\})\right)]
\end{aligned}$$

Here, we use z_i and w_i to emphasize that these are the extreme point solutions obtained at the ith LP. The first equality holds by construction of D_i. The second holds because $z_i(X,Y) = 1 - w_i(X,Y)$ and because for all $(X,Y) \in supp(D_{i-1})$, $OPT_{i-1}(X,Y) = OPT_i(X\cup\{(v_i,0)\}, Y\setminus\{(v_i,1)\})$ since $(v_i,0) \in OPT$.

If $(X,Y) \in supp(D_{i-1})$, then $X \subseteq OPT_{i-1}(X,Y)\setminus\{(v_i,0)\} \subseteq Y\setminus\{(v_i,1)\}$ and by submodularity:

$$g(OPT_{i-1}(X,Y)) - g(OPT_{i-1}(X,Y)\setminus\{(v_i,0)\}) \leq g(X\cup\{(v_i,0)\}) - g(X)$$

$$g(Y) - g(Y\setminus\{(v_i,1)\}) \leq g(OPT_{i-1}(X,Y)\setminus\{(v_i,0)\}\cup\{(v_i,1)\}) - g(OPT_{i-1}(X,Y)\setminus\{(v_i,0)\})$$

Adding these two inequalities, rearranging, and using the fact that $OPT_{i-1}(X,Y)\setminus\{(v_i,0)\}\cup\{(v_i,1)\} = OPT_i(X\cup\{(v_i,1)\}, Y\setminus\{(v_i,0)\})$ we obtain:

$$\begin{aligned}
g(OPT_{i-1}(X,Y)) \;-\; & g(OPT_i(X\cup\{(v_i,1)\}, Y\setminus\{(v_i,0)\})) \\
\leq\; & g(X\cup\{(v_i,0)\}) - g(X) - g(Y) + g(Y\setminus\{(v_i,1)\}) \\
=\; & f_i(X,Y)
\end{aligned}$$

Thus, we conclude:

$$\begin{aligned}
\mathrm{E}_{D_{i-1}}[g(OPT_{i-1}(X,Y)] \;-\; & \mathrm{E}_{D_i}[g(OPT_i(X,Y)] \\
\leq\; & \sum_{(X,Y)\in supp(D_{i-1})} w_i(X,Y)Pr_{D_{i-1}}[X,Y]f_i(X,Y) \\
=\; & \mathrm{E}_{D_{i-1}}[w_i(X,Y)f_i(X,Y)]
\end{aligned}$$

Analoguously, when $(v_i,1) \in OPT$ we obtain:

$$\mathrm{E}_{D_{i-1}}[g(OPT_{i-1}(X,Y)] - \mathrm{E}_{D_i}[g(OPT_i(X,Y)] \leq \mathrm{E}_{D_{i-1}}[z_i(X,Y)t_i(X,Y)]$$

On the other hand, we have:

$$\begin{aligned}\frac{1}{2}(\mathrm{E}_{D_i}[g(X) &+ g(Y)] - \mathrm{E}_{D_{i-1}}[g(X)+g(Y)]) = \frac{1}{2}(\mathrm{E}_{D_{i-1}}[z_i(X,Y)g(X\cup\{(v_i,0)\})\\ &+ w_i(X,Y)g(X\cup\{(v_i,1)\}) + z_i(X,Y)g(Y\setminus\{(v_i,1)\}) + w_i(X,Y)g(Y\setminus\{(v_i,0)\})\\ &- g(X)-g(Y)])\\ &= \frac{1}{2}(\mathrm{E}_{D_{i-1}}[z_i(X,Y)(g(X\cup\{(v_i,0)\}) + g(Y\setminus\{(v_i,1)\}) - g(X) - g(Y))\\ &+ w_i(X,Y)(g(X\cup\{(v_i,1)\}) + g(Y\setminus\{(v_i,0)\}) - g(X) - g(Y))])\\ &= \frac{1}{2}(\mathrm{E}_{D_{i-1}}[z_i(X,Y)f_i(X,Y) + w_i(X,Y)t_i(X,Y)])\\ &\geq \max\{\mathrm{E}_{D_{i-1}}[w_i(X,Y)f_i(X,Y)], \mathrm{E}_{D_{i-1}}[z_i(X,Y)t_i(X,Y)]\}\end{aligned}$$

where the first equality is by how D_i is constructed and the last inequality is because of inequalities 1 and 2 from the LP. □

To conclude the proof of the theorem, we add the inequalities given by the lemma for $1 \leq i \leq n$, obtaining:

$$\mathrm{E}_{D_0}[g(OPT_0(X,Y))] - \mathrm{E}_{D_n}[g(OPT_n(X,Y))] \leq \frac{1}{2}\left(E_{D_n}[g(X)+g(Y)] - E_{D_0}[g(X)+g(Y)]\right)$$

Notice that $\mathrm{E}_{D_0}[g(OPT_0(X,Y))] = g(OPT)$, $E_{D_0}[g(X)] = g(\emptyset)$, $E_{D_0}[g(Y)] = g(V\times\{0,1\})$, and for all $(X,Y) \in supp(D_n)$, $X = Y = OPT_n(X,Y)$. Therefore the inequality becomes

$$g(OPT) - \mathrm{E}_{D_n}[g(X)] \leq \frac{1}{2}\left(2E_{D_n}[g(X)] - g(\emptyset) - g(V\times\{0,1\})\right)$$

Therefore, after rearranging we get:

$$\mathrm{E}_{D_n}[g(X)] \geq \frac{1}{2}g(OPT) + \frac{1}{4}[g(\emptyset) + g(V\times\{0,1\})] \geq \frac{3}{4}g(OPT)$$

The last inequality follows from the fact that F is normalized (so $g(\emptyset) = 0$) and monotone (so $g(V\times\{0,1\}) \geq g(OPT)$). Calling the algorithm's ouput assignment A, we conclude that

$$F(A) = g\left(\operatorname*{argmax}_{(X,Y)\in supp_{D_n}}\{g(X)\}\right) \geq \mathrm{E}_{D_n}[g(X)] \geq \frac{3}{4}g(OPT) = \frac{3}{4}F(OPT)$$

We note that the LP format is the same as that in [16]. The only difference with their LP is the coefficients. So their argument that this can be solved by viewing it as a fractional knapsack problem still holds. □

4.2 Online width inapproximation bounds for max-2-sat

We now present width impossibility results for max-sat with respect to different input models. The best known efficient algorithm for max-sat has an approximation ratio of 0.797 [5]. Recall that Johnson's algorithm for max-sat [37] achieves a 2/3 approximation ratio [19] and only requires the algorithm to know the lengths of the clauses; i.e. input model 1. Even for input model 2, we show in Theorem 2 that exponential width-cut is required to improve upon the 3/4 approximation ratio

achieved by Algorithm 1, which is a linear width algorithm in input model 1. In Theorem 3, we show that constant width algorithms cannot achieve an approximation ratio of 2/3 in input model 0. This shows that constant width is unable to make up for the power lost if the algorithm does not know the lengths of clauses or if the algorithm is required to be deterministic; we note that the randomized algorithm using probabilities proportional to weights achieves an approximation ratio of 2/3 [6]. Finally, in Theorem 4 we show that, in input model 3, exponential width is required to achieve an approximation ratio greater than 5/6.

Our impossibility results hold even in some special cases of max-sat. In max-q-sat, the instance is guaranteed to have clauses of length at most q. Exact max-q-sat is the case where all clauses are of length exactly q. In the following theorem, we show that for input model 2, exponential width-cut cannot achieve a better approximation ratio than that achieved in Theorem 1, even for exact max-2-sat. It should be noted that a 3/4 approximation ratio is already achieved by the naive randomized algorithm (that sets a variable to 0 (or 1) with probability 1/2) and by its de-randomization, Johnson's algorithm, for exact max-2-sat.

We say that a max-of-k algorithm *assigns* (or *sets*) x to $(b_1, b_2, \ldots, b_k) \in \{0,1\}^k$ if it assigns x to b_1 in its first assignment, it assigns x to b_2 in its second assignment, etc.

Theorem 2. *For any $\epsilon > 0$ there exists $\delta > 0$ such that, for $k < e^{\delta n}$, no online width-cut-k algorithm can achieve an asymptotic approximation ratio of $3/4+\epsilon$ for unweighted exact max-2-sat with input model 2.*

Proof. First, we show a concrete example where any width-2 algorithm achieves an approximation ratio of at most 3/4. Then we show a way to extend this to a $3/4+\epsilon$ asymptotic inapproximation with respect to the max-of-k model for k that is exponential in the number of variables. Finally, we briefly argue why this impossibility result will also hold in the more general width-cut-k case.

Suppose $k = 2$. The adversary begins by showing variable x_1: it appears positively in one clause and negatively in another clause, both of length 2. The remaining variable in both clauses is y. If the algorithm does not branch or sets the variable x_1 to 0 or to 1 in both assignments, the adversary can force a 3/4 approximation ratio as follows. Suppose without loss of generality that in both assignments x_1 is assigned 1. Then the adversary presents the instance: $(x_1 \vee \overline{y}) \wedge (\overline{x_1} \vee y) \wedge (\overline{y} \vee z) \wedge (\overline{y} \vee \overline{z})$. No assignment where x_1 is set to 1 can satisfy all clauses, but an assignment where x_1 and y are set to 0 satisfies all clauses. Thus, the algorithm achieves an approximation ratio of at most 3/4.

Therefore the algorithm must set one assignment to 1 and the other to 0. We assume that the algorithm sets x_1 to $(1,0)$. Now the adversary presents a variable x_2, where again there is one clause where it appears positively and one where it appears negatively, both of length 2 and where the remaining variable is y. Then there are four cases depending on the decision of the algorithm on x_2 (in each case, the whole instance consists of four clauses in total):

Decision on x_2	$(0,0)$	$(1,0)$	$(1,1)$	$(0,1)$
Clause with x_1	$x_1 \vee \overline{y}$	$x_1 \vee \overline{y}$	$x_1 \vee \overline{y}$	$x_1 \vee y$
Clause with $\overline{x_1}$	$\overline{x_1} \vee \overline{y}$	$\overline{x_1} \vee y$	$\overline{x_1} \vee \overline{y}$	$\overline{x_1} \vee \overline{y}$
Clause with x_2	$x_2 \vee y$	$x_2 \vee y$	$x_2 \vee \overline{y}$	$x_2 \vee y$
Clause with $\overline{x_2}$	$\overline{x_2} \vee \overline{y}$	$\overline{x_2} \vee \overline{y}$	$\overline{x_2} \vee y$	$\overline{x_2} \vee \overline{y}$

In all cases, the algorithm will only be able to satisfy 3 out of the 4 clauses in any of its branches, but the instance is satisfiable, so the inapproximation holds.

Now let us show how to extend this idea to max-of-k. Let $\epsilon > 0$, take $\delta = 7\epsilon^2$, so that $k < e^{\delta n} = e^{7\epsilon^2 n}$. The adversary will present variables $x_1, \ldots, x_n$, where each x_i appears in two

clauses of length 2 and where the remaining variable is y: in one x_i appears positively and in the other negatively. In fact, the two clauses will either represent an equivalence to y (given by $x_i \vee \overline{y}$, $\overline{x_i} \vee y$) or an inequivalence to y (given by $x_i \vee y$, $\overline{x_i} \vee \overline{y}$), but the algorithm does not know which is the case. If an assignment does not satisfy the (in)equivalence correctly, it will get only one of the two clauses (ie 1/2 of the total).

Suppose that the algorithm maintains k assignments, and suppose it makes assignments on $x_1, \ldots, x_n$. Then the algorithm can only maintain at most k of the possible 2^n assignments. For a fixed assignment of $x_1, \ldots, x_n$, by Chernoff bounds, the probability that a uniformly random assignment agrees with the fixed one on at least $n/2 + 2\epsilon n$ variables is at most $e^{-8\epsilon^2 n}$. Similarly, the probability that it agrees with the fixed assignment on at most $n/2 - 2\epsilon n$ variables is at most $e^{-8\epsilon^2 n}$. Thus, by union bounds, the probability that any of the two possibilities occurs on any of the k assignments maintained by the algorithm is at most $2ke^{-8\epsilon^2 n} < e^{-\epsilon^2 n + ln2} < 1$. So there exists an assignment A that agrees with every assignment maintained by the algorithm on more than $n/2 - 2\epsilon n$ but less than $n/2 + 2\epsilon n$ of the variables.

The adversary uses this assignment A to determine the signs of y in the clauses, which in turn determines for each i whether x_i is equivalent or inequivalent to y. If A assigns x_i to 1, then the adversary says x_i is equivalent to y. If A assigns x_i to 0, then the adversary says that x_i is inequivalent to y. Clearly, the set of clauses constructed is satisfiable. Fix one of the k assignments maintained by the algorithm. If to complete this assignment the algorithm sets y to 1, the number of (in)equivalences satisfied by the assignment is equal to the number of variables where A and this assignment agree, which is less than $n/2 + 2\epsilon n$. On the other hand, if to complete the assignment the algorithm sets y to 0, then the number of (in)equivalences satisfied is equal to the number of variables where A and this assignment disagree, which again is less than $n/2 + 2\epsilon n$. Since an assignment that satisfies q (in)equivalences will satisfy a $\frac{n+q}{2n}$ fraction of the clauses, the approximation ratio achieved by the algorithm is less than $3/4 + \epsilon$.

It is easy to see why this result will also hold for width-cut: the only decisions of the adversary that depend on the branching are made when the last variable y is being processed (their signs are determined in each clause by assignment A). So the adversary can use the strategy that corresponds to the assignments of the algorithm right before y is presented. Any branching or cutting made when deciding the assignments for y are irrelevant: since it is the last variable, the algorithm should just assign y to maximize the number of satisfied clauses in each assignment. □

Theorem 3. *For any constant k, the asymptotic approximation ratio achieved by any online width-k algorithm for unweighted max-sat with input model 0 is strictly less than* 2/3*.*

Proof. We start by giving a max-of-k inapproximation result, which is then easily extended to width. It should be noted that for this result we need to allow the adversary's final instance to contain repeated equal clauses.

First consider the case $k = 2$. The adversary presents a variable x_1. There are two clauses where it appears positively and two where it appears negatively. Without loss of generality, there are two options: both assignments set x_1 to 1 or the first sets x_1 to 1 and the second sets x_1 to 0. In the former case, the adversary proceeds to say that the clauses containing $\overline{x_1}$ were of length one but the clauses containing x_1 had an additional variable y, which means both assignments satisfy half of the clauses. In the latter case, the adversary now presents variable y_1. It appears positively in one of the clauses where $\overline{x_1}$ appears and it appears negatively in the other clause where $\overline{x_1}$ appears. The value the second assignment gives to this variable is irrelevant since it

already satisfied these clauses. Without loss of generality, assume the first assignment sets it to 1. Then the adversary presents a variable z_1, which occurs only positively in the clause where y_1 appears positively. Thus the values both assignments give to this variable are irrelevant. The first assignment satisfied three of the four clauses while the second only satisfied two of them. There is an optimal solution satisfying all four: set x_1 to 1, y_1 to 0, z_1 to 1. Now, the adversary repeats this process, but reversing the roles of the two assignments so that now the first assignment only gets two out of four clauses and the second gets three. Adding up, both assignments get five out of the eight clauses and the optimal value is 8, so we get a 5/8 inapproximation.

For the general (but constant) k case, we proceed by induction to prove that there is an adversary giving an inapproximation ratio strictly less than 2/3. Recall that online maxsat ($k = 1$) in this model cannot get an approximation ratio better than 1/2 [6] (it is easy to extend this to an asymptotic inapproximation). Suppose there is an adversarial strategy for $i < k$. We now present a strategy for max-of-k. The adversary begins by presenting many variables $v_1, v_2, \ldots, v_n$, each of which will have many clauses where it appears positively and the same number of clauses where it appears negatively. For each $1 \leq i \neq j \leq n$, the clauses where v_i and v_j appear are disjoint. These will be all of the clauses of the instance: the adversary will not present any new clauses later on. It will only present additional variables contained within these clauses.

After decisions are made there will be k assignments, each assigning a value 0 or 1 to each of the variables. For every $S \in \{0,1\}^k$, the adversary will recursively apply its strategies for max-of-i (for values $i < k$) to V_S, the set of variables from $v_1, v_2, \ldots, v_n$ assigned to S, and to the clauses where variables in V_S appear. More precisely, let $\mathbb{A}_j$ be the adversary for max-of-j. Then $\mathbb{A}_k$ will simulate $\mathbb{A}_i$ by ignoring some of the assignments (since now it can only consider i of them). Variables created by $\mathbb{A}_i$ will be new variables. When $\mathbb{A}_i$ creates new clauses, $\mathbb{A}_k$ uses some of the clauses where variables in V_S appear instead of creating new ones. If $S = (1, \ldots, 1)$, for these variables the adversary says that the clauses where they appear positively have length two and include a new variable y but the clauses where they appear negatively have length one. Thus the algorithm can only satisfy 1/2 of these clauses but the set of clauses is satisfiable. The response is analogous if $S = (0, \ldots, 0)$.

Now suppose that $S = (s_1, \ldots, s_k)$ contains t 1's and $k - t$ 0's, for some for $0 < t < k$. Let $S_1 := \{1 \leq j \leq k | s_j = 1\}$ and let $S_0 := \{1 \leq j \leq k | s_j = 0\}$, so $|S_1| = t$ and $|S_2| = k - t$. The adversary will roughly split V_S into two parts, one V_S^1 of size $w|V_S|$ and the other V_S^2 of size $(1 - w)|V_S|$, for a w to be determined. In the first part, the adversary will say that the positive clauses (where variables in V_S^1 appear positively) were of length one, and it will simulate $\mathbb{A}_t$ using the negative clauses, so the decisions made on the $k - t$ assignments indexed by S_0 don't matter and the adversary only considers the decisions of the algorithm on the t assignments indexed by S_1. In the second part, the adversary will say that negative clauses were of length one, and will simulate $\mathbb{A}_{k-t}$ using the positive clauses and considering only the $k - t$ assignments indexed by S_0. See Table 1 for an example. Let r_1 be the inapproximation ratio for max-of-t and let r_2 be the inapproximation ratio for max-of-$(k - t)$. Let $a = (1 + r_1)/2$ and $b = (1 + r_2)/2$. Then the proportion of clauses satisfied by assignments indexed by S_1 is at most $aw + 1/2(1 - w)$ and the proportion of clauses satisfied by assignments indexed by S_0 is at most $1/2w + b(1 - w)$. We select w to minimize the maximum between these two amounts, by equating these values:

$$aw + \frac{1}{2}(1 - w) = \frac{1}{2}w + b(1 - w)$$

We solve this equation, obtaining

$$w = \frac{b - \frac{1}{2}}{a + b - 1}$$

Plugging back in into the equality and considering there is an optimal assignment satisfying all clauses, we obtain an inapproximation ratio of

$$\frac{2ab - \frac{1}{2}}{2(a + b - 1)}$$

V_S^1		V_S^2	
v_1	Clause of length 1	v_4	Used to simulate $\mathbb{A}_{k-t}$
v_2	Clause of length 1	v_5	Used to simulate $\mathbb{A}_{k-t}$
v_3	Clause of length 1	v_6	Used to simulate $\mathbb{A}_{k-t}$
$\overline{v_1}$	Used to simulate $\mathbb{A}_t$	$\overline{v_4}$	Clause of length 1
$\overline{v_2}$	Used to simulate $\mathbb{A}_t$	$\overline{v_5}$	Clause of length 1
$\overline{v_3}$	Used to simulate $\mathbb{A}_t$	$\overline{v_6}$	Clause of length 1

Table 1: Example where V_S consists of 6 variables, 3 in V_S^1 and 3 in V_S^2.

Now, recall r_1, r_2 are inapproximation ratios for max-of-k' for $k' < k$, so by the induction hypothesis $1/2 \leq r_1, r_2 < 2/3$, which implies $3/4 \leq a, b < 5/6$. Given these parameters, it can be shown that the above ratio gets a value strictly less than $2/3$. Since all ratios are less than $2/3$ regardless of S, the inapproximation obtained overall is less than $2/3$. Notice that we assume that we can neglect the gain obtained when $|V_S|$ is not large enough to apply the recursive strategy (hence the number of initial variables n has to be large) and in addition we are assuming we can at least approximate w accurately (hence the number of clauses per v_i has to be large).

To extend this result to width k, we can begin by assuming that $k' = 1$ where k' is the maximum number of assignments the algorithm keeps. We start by applying the adversary for max-of-k'. If the adversary finishes before the algorithm does any splitting then we are done. Otherwise the algorithm splits to now maintain $k'' > k'$ assignments and we apply the adversary for max-of-k'', but using many more clauses so that we ensure that the clauses used by previous adversaries will be negligible when calculating the approximation ratio. □

Theorem 4. *For any $\epsilon > 0$ there exists $\delta > 0$ such that, for $k < e^{\delta n}$, no online width-cut-k algorithm can achieve an asymptotic approximation ratio of $5/6 + \epsilon$ for unweighted max-2-sat with input model 3.*

Proof. We use an argument similar to the one in Theorem 2, but with a different clause construction. Given $\epsilon > 0$, let $\delta = 144\epsilon^2$ so that $k < e^{\delta n} = e^{144\epsilon^2 n}$. The instance will contain variables $x_1, \ldots, x_{n/2}$ and $y_1, \ldots, y_{n/2}$. The clauses will be $x_i \vee \overline{y_i}$, $\overline{x_i} \vee y_i$, and either y_i or $\overline{y_i}$. The first two clauses are satisfied if and only if $x_i = y_i$. The last clause determines whether y_i should be assigned to 0 or 1. Any such set of clauses will be satisfiable. The adversary presents the x_i's, and the width-cut algorithm produces at most k assignments of these variables.

For a fixed assignment of the $\frac{n}{2}$ x_i's, the probability that a uniformly random assignment agrees with the fixed one on more than $n/4 + 6\epsilon n$ of the variables is at most $e^{-144\epsilon^2 n}$. Therefore, there exists an assignment A that agrees with each of the assignments maintained by the algorithm on

at most $n/4 + 6\epsilon n$ variables. The adversary now presents the y_i's. It chooses to include clause y_i in the instance if x_i is set to 0 in assignment A, and it includes $\overline{y_i}$ if x_i is set to 1.. Whenever an assignment does not agree with A on x_i, it will satisfy at most two of the three clauses where y_i appears in. Therefore, no assignment can satisfy more than a $\frac{1(1/4+6\epsilon)+2/3(1/2-(1/4+6\epsilon)))}{1/2} = 5/6 + \epsilon$ fraction of the clauses. □

5 Bipartite matching results

We first consider inapproximation results showing that in the max-of-k parallel model, k must be greater than $\frac{\log n}{\log \log n}$ to asymptotically improve upon the $\frac{1}{2}$ approximation given by any maximal matching algorithm. We then consider bipartite matching in the priority and ROM models. Our priority inapproximation shows that the randomization cannot be replaced by a judicious but deterministic ordering of the online vertices.

5.1 Width inapproximation

We first fix some notation for max-of-k online bipartite matching. The algorithm keeps k distinct matchings $M_1, \ldots, M_k$. Whenever an online vertex u arrives, it can update each of the M_j's by matching to u one of its neighbours that M_j has not yet matched. The size of the matching obtained by the algorithm is the maximum size of the M_j's. We assume that the online vertices are numbered from 1 to n, and the algorithm receives them in that order. The adversary chooses the offline vertices that are the neighbours. We will refer to the time when the algorithm chooses the matchings for the i-th online vertex as step i.

In the usual online bipartite matching problem, we can assume that the algorithm is greedy in the sense that when an online node arrives, it will be matched if it currently has any unmatched offline neighbors. Note that there can be many greedy algorithms depending on the way in which an algorithm chooses amongst different unmatched neighbors. To see that we can assume greediness for online bipartite matching, we argue as follows. Whenever, a non greedy algorithm is about to pass up a match, we now force an arbitrary match to say some node v. Later, if another online node u wishes to match with v, it will choose another match if possible, and if not possible there is no loss in the total number of matched nodes by having forced the greedy match ealier. This argument clearly still applies when we keep track of multiple matchings at once: we can assume that the algorithm is greedy in each. We begin with a statement of a max-of-n^k algorithm and inapproximability result that follows immediately from the relationship between advice and max-of-k algorithms:

Theorem 5. *For every $\epsilon > 0$ there exists a max-of-$n^{O(1)}$ algorithm achieving an approximation ratio of $1 - 1/e - \epsilon$. Also, no max-of-$2^{o(n)}$ algorithm can achieve an approximation ratio better than $1 - 1/e + \epsilon$.*

Proof. By Böckenhauer et al [11], as observed in [23], for every $\epsilon > 0$ there is a $\Theta(\log n)$ advice algorithm achieving an approximation ratio of $1 - 1/e - \epsilon$. The algorithm is a de-randomization of the Ranking algorithm. Part of the advice string consists of an encoding of n. Even without this, the advice is still $\Theta(\log n)$. By Lemma 1 there is a max-of-$2^{O(\log n)} = n^{O(1)}$ algorithm getting the desired approximation ratio. It should be noted that the algorithm uses an information-theoretic approach and is, in fact, extremely inefficient, in addition to requiring heavy pre-processing.

For the other part of the theorem, Mikkelsen [45] showed that, for every ϵ, an advice algorithm with approximation ratio $1 - 1/e + \epsilon$ requires $\Omega(n)$ advice. If there was a max-of-$2^{o(n)}$ with this ratio, then by Lemma 1 there would be a $o(n)$ advice algorithm achieving that ratio. Note that it can be assumed without loss of generality that any algorithm with $\Omega(\log n)$ advice knows n. □

We now prove some impossibility results concerning algorithms trying to beat the 1/2 barrier that deterministic online algorithms cannot surmount. The adversarial graphs will be bipartite graphs with perfect matchings. The adversary will not only provide the graph but also construct a perfect matching "online". Once an offline vertex has been used in the adversary's perfect match, the adversary will not present it as a neighbour of any of the remaining online vertices. When an online vertex u arrives, the adversary will choose a nonempty subset of offline vertices as its set of neighbours. Then the algorithm (which we assume without loss of generality to be greedy) chooses a match in each of the k matchings. For each matching M_i, if there are neighbours of u that have not been used in M_i, the algorithm must pick a neighbour v and match u to v in M_i. When the algorithm has finished making its choices, the adversary picks one of the neighbours of u and adds the pair of vertices to the perfect matching that it is constructing. The match to online vertex i in this perfect matching is labelled as offline vertex i. We say that this offline vertex becomes *unavailable*. An offline vertex is *available* if it is not unavailable. The goal of the adversary is to force the algorithm to make as few matches as possible in the best of its k matchings.

At a specific point in time and for any offline vertex v, we say $t(v)$ is the number of the algorithm's matchings that have used v. Whenever we say that the adversary *gets rid* of an offline vertex v at a given step we mean that, at this step, the only neighbour of the online vertex u is v, so the best option of the algorithm is to match u to v in any of the matchings where v has not yet been used. Also, v will not be a neighbour of any of the remaining online vertices (the adversary must add (u, v) to its perfect matching). If the adversary only gets rid of a constant number of offline vertices, the matchings made by the algorithm during these steps are negligible: they do not affect the asymptotic approximation ratio.

Lemma 3. *For any max-of-2 online bipartite matching algorithm, there exists an n by n bipartite graph with a perfect matching such that the algorithm obtains a matching of size at most $n/2 + 3$.*

Proof. Since the algorithm has only two matchings, at any point in time and for any offline vertex v, $0 \leq t(v) \leq 2$. There will be two stages. The first stage consists of steps where, at the beginning of the step, there are more than two available vertices v with $t(v) = 0$. The adversary chooses as neighbours all available vertices. Thus, we can guarantee that, after the algorithm has chosen matches for the online vertex of this step, there will still be at least one available vertex v with $t(v) = 0$. The adversary will choose one such offline vertex (i.e. one not chosen in any matching) to be added to its perfect matching. If at the beginning of a step, there are less than 3 available offline vertices with $t(v) = 0$ (so that we cannot guarantee that there will exist a vertex with $t(v) = 0$ after the algorithm does its matches), the adversary concludes stage 1 and gets rid of the at most 2 available offline vertices with $t(v) = 0$ before stage 2.

Let p be the number of steps that occurred during stage 1. Let p_1 and p_2 be the number of offline vertices with $t(v) = 1$ and $t(v) = 2$ at the end of stage 1, respectively. During stage 1, at each step, $\sum_v t(v)$ is incremented by 2, so $2p = p_1 + 2p_2$. (It follows that p_1 is an even number.) At each step in stage 1 a vertex becomes unavailable, but no vertex with $t(v) \neq 0$ becomes unavailable, so $n \geq p + p_1 + p_2$ (inequality because the adversary may get rid of vertices). Therefore, $p = \frac{p_1}{2} + p_2$

and $\frac{n}{2} \geq \frac{3}{4}p_1 + p_2$. Right before the beginning of stage 2, the size of each matching is at most $p+2$: p from stage 1 and 2 from getting rid of vertices.

The second stage consists of steps where at the beginning of the step all the available offline vertices satisfy $t(v) \geq 1$, and there are some available vertices with $t(v) = 1$. At this stage, the p_1 vertices with $t(v) = 1$ are considered. Since the number of vertices matched in M_1 and M_2 have to be the same during stage one, half of the vertices are used in M_1 and half in M_2. The steps in this stage will be either M_1-steps or M_2-steps (the order in which the adversary does them is irrelevant). An M_1 step is a step where the neighbours of the online vertex are available vertices that have already been matched in M_1, so the algorithm can only do a match in M_2. After the algorithm does its match, if there is still an available vertex that has only been matched in M_1, the adversary adds it to the perfect matching it is constructing, making it unavailable. At the beginning of stage 2, there are $p_1/2$ vertices with $t(v) = 1$ matched in M_1. Every M_1-step (except possibly the last, where there may be only one available vertex matched in M_1 with $t(v) = 1$) makes two vertices unusable: one because it is made unavailable and one because $t(v)$ changes from 1 to 2. So there will be $\lceil p_1/4 \rceil$ M_1-steps. We define M_2-steps analogously, and there will be $\lceil p_1/4 \rceil$ M_2-steps. After stage 2, all available vertices have $t(v) = 2$ and are unusable, so no more matchings are made by the algorithm. The adversary can finish the construction of the perfect matching by making the remaining online vertices be neighbours of all offline vertices with $t(v) = 2$.

The size of the matchings produced by the algorithm is at most $p+2+\lceil \frac{p_1}{4} \rceil$, since the algorithm can only increase the size of M_1 during stage 1, while the adversary gets rid of vertices, and during M_2-steps, and similarly for M_2. But $p + 2 + \lceil \frac{p_1}{4} \rceil \leq p + 3 + \frac{p_1}{4} = \frac{p_1}{2} + p_2 + 3 + \frac{p_1}{4} \leq \frac{n}{2} + 3$. This in particular implies that the asymptotic approximation ratio achieved by any *max-of-2* online algorithm is at most $\frac{1}{2}$. □

Clearly, making the width bigger without making n bigger will eventually allow the algorithm to obtain an optimal matching using brute-force. However, it is natural to wonder whether by allowing n to be large the adversary will be able to trick the algorithm into producing a small matching. This question is answered by the following:

Theorem 6. *For any constant k, any width-k online bipartite matching algorithm cannot achieve an asymptotic approximation ratio greater than $\frac{1}{2}$.*

Proof. We prove the theorem by first considering max-of-k and then extending the result to width. We prove the following statement by induction on k: for any max-of-k algorithm there exists a constant c_k (that only depends on k) such that for every n there is a graph G of size n (n vertices on each of the two sides) that has a perfect matching but the algorithm only obtains a matching of size at most $n/2 + c_k$. For $k = 1$, the problem is the well studied online bipartite matching problem: there are adversarial graphs where we can take $c_1 = 1$ (needed for odd values of n). Lemma 3 proves the case for $k = 2$, taking $c_2 = 3$. We assume that the claim is true for max-of-i for all $1 \leq i \leq k$, and then prove it for max-of-$(k + 1)$. As before, the adversary will decide the neighbours of the incoming online vertex as well as the offline vertex that matches it in the perfect matching it constructs (and this offline vertex will not be a neighbour of any of the remaining online vertices). Let $M_1, \ldots, M_{k+1}$ be the matchings that the algorithm constructs.

The adversary will have the same first stage as in the lemma, at each step adding to its perfect matching a vertex with $t(v) = 0$. When there are less than $k + 2$ available vertices with $t(v) = 0$ (so we cannot guarantee that there will be an available vertex with $t(v) = 0$ after the algorithm does its matchings), stage 1 ends and then the adversary gets rid of all the available vertices with

$t(v) = 0$. Let p be the number of steps in the first stage and let q be the number of available offline vertices right after stage 1; that is, the number of offline vertices not in the adversary's perfect matching at that time. Then $p + q = n$, and after the adversary gets rid of vertices there are at most q available vertices. Now the adversary proceeds to a second stage.

For every $S \subseteq \{1, \ldots, k+1\}$ with $S \neq \emptyset$ and $S \neq \{1, \ldots, k+1\}$, let V_S be the subset of available offline vertices that have been used in M_i for all $i \in S$ and that have not been used in M_i for all $i \notin S$. Let $S^C = \{1, \ldots, k+1\} \setminus S$, and notice that $0 < |S^C| < k+1$. At this point, only M_i's with $i \in S^C$ can match vertices in V_S. The idea is that we recursively apply our adversary for max-of-$|S^C|$ algorithms on a graph with V_S as the set of offline vertices and with $|V_S|$ online vertices. The M_i's with $i \in S$ are ignored: the algorithm cannot add matches in these when the set of neighbours of the online vertex is a subset of V_S. By the induction hypothesis, there is an adversarial strategy for max-of-$|S^C|$ such that the size of any of the matchings obtained (on a graph that uses $|V_S|$ online vertices and V_S as the offline vertices) is at most $|V_S|/2 + c_{|S^C|}$. This will be close to 1/2 of the total when $|V_S|$ is large, since $c_{|S^C|}$ is a constant.

In the second stage, the adversary executes the max-of-$|S^C|$ strategies described above. For $S_1 \neq S_2$, the strategies will be independent because the set of offline neighbours is disjoint. Thus, the order in which the strategies are executed is irrelevant: they could even be executed in parallel. For concreteness, suppose the adversary first executes the strategies for subsets S of size k in lexicographic order (here it applies max-of-1 strategies), then for subsets S of size $k-1$ in lexicographic order (here it applies max-of-2 strategies), etc. After the strategies for all subsets have been executed, stage 2 is concluded and now we need to show that the adversary's perfect matching is about twice the size as any of the matchings constructed by the algorithm.

For simplicity, ignore the adversary getting rid of vertices and suppose that every V_S is large enough. In the end, any fixed matching M_i will use p offline vertices because of stage 1. After applying our recursive adversaries, M_i will use roughly half of the offline vertices that were not used by M_i by the end of stage 1 but were still available at this time. The number of offline vertices that are available by the end of stage 1 is q. Thus, in the end, the size of matching M_i is $p + \frac{q-p}{2} = \frac{p+q}{2} = n/2$.

Now we make the intuition from the previous paragraph precise. Notice that the size of matching M_i is at most $p + k + 1 + \sum_{S: i \notin S} \frac{|V_S|}{2} + c_{|S^C|}$: p during stage 1, $k+1$ from vertices the adversary gets rid of before stage 2, and the rest during stage 2. The number of available offline vertices that M_i has not used at the beginning of stage 2 is $\sum_{S: i \notin S} V_S \leq q - p = n - 2p$. By the induction hypothesis, $\sum_{S: i \notin S} c_{|S^C|} \leq \sum_{1 \leq j \leq k} \binom{k}{j-1} c_j$, since there are $\binom{k}{j-1}$ ways of choosing S^C of size j if we require $i \in S^C$. Therefore, the size of any matching obtained by the algorithm is at most $\frac{n}{2} + c_{k+1}$ where $c_{k+1} = k + 1 + \sum_{1 \leq j \leq k} \binom{k}{j-1} c_j$. This concludes the induction and the proof for max-of-k algorithms.

Now we extend the result to width k. The idea is that we slightly modify the adversary so that, given the decisions of the algorithm, for $a > b$, a max-of-a adversary can be viewed as a max-of-b adversary. The width-k adversary will use this fact to change from the max-of-b adversary to the max-of-a adversary, whenever the algorithm branches, without affecting the argument. Let $\mathbb{A}_j$ be the max-of-j adversary, but where the condition to end stage 1 is that there are less than $k+1$ available vertices with $t(v) = 0$, instead of $j+1$. Also, we assume that $\mathbb{A}_j$ may perform the independent stage 2 simulations in any order we choose. The width-k adversary $\mathbb{A}$ does the following: begin by assuming m, the maximum number of matchings maintained by the algorithm, is 1. When $\mathbb{A}$ needs to tell the algorithm which are the neighbours of the next online vertex, $\mathbb{A}$

does whatever $\mathbb{A}_m$ would do given the matchings the algorithm has made so far. If the algorithm does not branch, $\mathbb{A}$ constructs the perfect match as $\mathbb{A}_m$ would, and this finishes the processing of the online vertex. On the other hand, the algorithm may branch on the decisions of the online vertex, so that now it maintains $m + r$ matchings. Each new matching M_{new} will branch off of some matching M_{old}, which in the branching tree means that now M_{new} is a leaf of the subtree rooted at M_{old} (or any of its ancestors). In this case, $\mathbb{A}$ simply increases m by r. Then it simulates $\mathbb{A}_m$ (m is the increased value) to obtain the perfect match. And this finishes the processing of the online vertex.

At any point in time, there is a max-of-m algorithm that simulates the width-k algorithm up to this point, if it knows the branching tree created up to this point. For each level of the branching tree (each corresponding to an online vertex), the max-of-m algorithm keeps l copies of each node, where l is the number of leaves in the subtree rooted at that node. We claim that all previous decisions made by $\mathbb{A}$ are consistent with $\mathbb{A}_m$, in the following sense: the behaviour of $\mathbb{A}$ on the width-k algorithm (which so far only has m branches) is equivalent to the behaviour of $\mathbb{A}_m$ on the max-of-m algorithm just described. By behaviour of an adversary, we mean the offine neighbours it presents and the perfect matchings it constructs at each step.

We can show this by induction. Consider a step where the width-k algorithm branches, and let m_i and m_f be the values of m at the beginning and at the end of the step, respectively. Suppose that $\mathbb{A}$ (on the width-k algorithm) behaves as $\mathbb{A}_{m_i}$ on the max-of-m_i algorithm that simulates the width-k algorithm up until the previous step. We now show that $\mathbb{A}$ behaves as $\mathbb{A}_{m_f}$ on the max-of-m_f algorithm that simulates the width-k algorithm up until this step. On later steps, as long as the algorithm does not branch, this consistency will still hold. We see $\mathbb{A}$ as $\mathbb{A}_{m_i}$ (on the max-of-m_i algorithm) up until the end of the previous step, which is valid by our assumption. If the branching occurs during stage 1, then what we claim is true since $\mathbb{A}_{m_i}$ and $\mathbb{A}_{m_f}$ have the same stage 1.

Now suppose that the branching occurs during stage 2. We prove that up until the beginning of the current step we can make the behaviour of $\mathbb{A}_{m_f}$ on the max-of-m_f algorithm be the same as the behaviour of $\mathbb{A}_{m_i}$ on the max-of-m_i algorithm. On a stage 2 step, $\mathbb{A}_{m_i}$ will be simulating $\mathbb{A}_x$ on V_S for some $S \subseteq \{1, \ldots, m_i\}$ and where $x = |S^C|$. Then in $\mathbb{A}_{m_f}$ we choose to simulate a step of $\mathbb{A}_y$ on $V_{S'}$, where $S \subseteq S' \subseteq \{1, \ldots, m_f\}$, S' contains all indices of matchings that branched off from matchings indexed in S, and $y = |S'^C| \geq |S^C|$ (S'^C contains indices of matchings that branched off from matchings indexed in S^C). This is because by definition the max-of-m_f algorithm only keeps copies of the matchings that will later branch off, so $V_S = V_{S'}$: here the left hand side corresponds to the set according to $\mathbb{A}_{m_i}$ and the right hand side is according to $\mathbb{A}_{m_f}$. More generally, there is a mapping $f : \mathcal{P}(\{1, \ldots, m_i\}) \to \mathcal{P}(\{1, \ldots, m_f\})$ that maps a set of indices Q to the set of indices of matchings that branch off from matchings indexed in Q. Because of the behaviour of the algorithms, it holds that V_Q according to $\mathbb{A}_{m_i}$ is equal to $V_{f(Q)}$ according to $\mathbb{A}_{m_f}$, and for any $R \subseteq \{1, \ldots, m_f\}$ that does not have a preimage under f, $V_R = \emptyset$. In the step where m changes from m_i to m_f, $\mathbb{A}_{m_i}$ simulating $\mathbb{A}_x$ to select a subset of V_S as the set of neighbours is equivalent to $\mathbb{A}_{m_f}$ simulating $\mathbb{A}_y$ to select a subset of $V_{S'}$. After the algorithm does its decision and branching, m is updated and $\mathbb{A}$ actually simulates $\mathbb{A}_{m_f}$, so the behaviour is the same. This concludes the proof of our claim.

Thus, in the end, the behaviour of $\mathbb{A}$ on the width-k algorithm is equivalent to the behaviour of $\mathbb{A}_k$ on the max-of-k algorithm. This means that the size of the matching constructed by the width-k algorithm is at most $n/2 + c_k$. Since we changed stage 1 of the adversaries, the c_k's will be somewhat larger, but they still only depend on k. □

Corollary 1. *Let $t(n) = \frac{\log n}{\log\log n}$. Any max-of-$t(n)$ online bipartite matching algorithm cannot achieve an asymptotic approximation greater than $\frac{1}{2}$.*

Proof. For any k, from the proof of Theorem 6 max-of-k algorithms can achieve matchings of size at most $n/2 + c_k$ on some hard graphs. First, we note that $c_k \leq k^k$. This is true for $k = 1$. Assuming this holds for $1 \leq i < k+1$, then $c_{k+1} \leq k+1+\sum_{1\leq j\leq k}\binom{k}{j-1}j^j < \sum_{0\leq j\leq k+1}\binom{k+1}{j}k^j = (k+1)^{k+1}$. In the second inequality we use the fact that $k+1 \leq 1+k^{k+1}$.

Now, notice that $t^t < (\log n)^{\frac{\log n}{\log\log n}} = (2^{\log\log n})^{\frac{\log n}{\log\log n}} = n$, where in the first inequality we omit dividing by $(\log\log n)^{\frac{\log n}{\log\log n}}$. This means that $t^t = o(n)$. Thus, max-of-t algorithms achieve matchings of size at most $n/2 + t^t = n/2 + o(n)$. □

The following corollary follow immediately from the observations in Section 2.2.3. Dürr et al [23] proved an $\Omega(\log\log\log n)$ advice lower bound for achieving an approximation ratio greater than $1/2$. Their advice lower bound applied to a natural but somewhat restricted class of online advice algorithms. We improve this result:

Corollary 2. *$\Omega(\log\log n)$ advice is required for an online algorithm to achieve an asymptotic approximation ratio greater than $1/2$ for bipartite matching, even when the algorithm is given n in advance.*

Proof. No $\log\left(\frac{\log n}{\log\log n}\right) = \log\log n - \log\log\log n$ advice algorithm can achieve an asymptotic approximation ratio better than $1/2$, even knowing n. Otherwise Lemma 1 would give a max-of-$\frac{\log n}{\log\log n}$ online algorithm achieving this ratio, contradicting the previous corollary. □

5.2 Priority Inapproximation Bounds

We now turn to study unweighted bipartite matching in the priority model. All of our results are for adaptive priority. While there are two related models where priority bipartite matching may be studied: one-sided (where there is one data item per vertex in the online side, and the offline side is known in advance) and two-sided (where there is one data item per vertex in the graph), we shall restrict attention to the one sided model.

5.2.1 The inapproximation for deterministic priority algorithms

The following theorem shows that deterministic priority algorithms cannot achieve a non-trivial asymptotic approximation ratio in the one-sided model. Thus, being able to choose the order in which to process the vertices is not sufficient to overcome the lack of randomness.

Theorem 7. *There does not exist a deterministic priority algorithm that achieves an asymptotic approximation ratio greater than $1/2$ for online one-sided bipartite matching, even if the algorithm knows the size of the graph.*

Proof. We describe a game between an algorithm and an adversary that, for any odd integer $n \geq 3$, yields an inapproximation of $\frac{(n+1)/2}{n}$, which can be made arbitrarily close to $1/2$ by making n sufficiently large. The bipartite graph will have two sides, each with n vertices. Let OFF be the set of offline vertices and let ON be the set of online vertices. We restrict our attention to graphs in which all vertices in ON have degree $(n+1)/2$. The idea is that the algorithm does not know, a

priori, anything about the degrees of vertices in OFF, so we can adjust the neighbours of vertices in OFF to ensure that the algorithm makes mistakes at each step.

The adversary will keep track of M, U, R, which are pairwise disjoint subsets of OFF (all initially empty). M will be the set of vertices in OFF matched by the algorithm, U will be the set of vertices that the algorithm cannot possibly match because of the matches it has already done, and R will be the set of vertices that the algorithm cannot match because of the rejections it has made. Note that the algorithm won't be able to match vertices in both U and R. The only difference between the sets is the reason for this "unmatchability". As the game between the algorithm and adversary progresses, the adversary announces some information about the graph, so that the set of possible instances may be further restricted.

The adversary will ensure that $|M| = |U|$ whenever the algorithm has to provide an ordering of data items. The set P of potential data items is defined as the set of vertices of degree $(n+1)/2$ where the set of neighbours N satisfies $M \subseteq N$ and $N \subseteq OFF \setminus (U \cup R)$. In other words, N contains M and is disjoint from U and R. Initially P consists of all data items of vertices of degree $(n+1)/2$ with neighbours in OFF, and P shrinks every time M, U, and R are updated. While $|U| + |R| < (n-1)/2$, the algorithm receives the data item of the vertex v from P that comes first in some ordering π of data items. Note that the number of neighbours in $OFF \setminus (U \cup R \cup M)$ is at least 2 because $|M| < (n-1)/2 = (n+1)/2 - 1$. There are two options: the algorithm matches v to a neighbour (in $OFF \setminus (U \cup R \cup M)$, because vertices in M have already been matched by the algorithm), or it rejects v. We now show how we maintain our invariant ($|M| = |U|$) in either case.

If the algorithm matches v to some vertex $m \in OFF \setminus (U \cup R \cup M)$, the adversary updates M by adding m. This means that all the vertices in ON that have not been processed yet will be neighbours of m. It picks another neighbour u of v from the set $OFF \setminus (U \cup R \cup M)$ and updates U by adding u. This implies that none of the vertices in ON that have not been processed yet will be neighbours of u, so it is impossible for the algorithm to match vertex u. If the algorithm rejects v, the adversary updates R by adding a neighbour r of v from $OFF \setminus (U \cup R \cup M)$. Thus, none of the vertices in ON that have not been processed yet are neighbours of r and r will remain unmatched. Note that, in either case, the condition $|M| = |U|$ is still maintained.

Each time a vertex in ON is examined, exactly one of $|U|$ and $|R|$ is increased by 1. Thus, after $(n-1)/2$ vertices in ON have been examined, $|U| + |R| = (n-1)/2$. At this point, P consists of vertices whose set of neighbours is $OFF \setminus (U \cup R)$. This is necessary to guarantee that the potential data items correspond to vertices of degree $(n+1)/2$. The adversary no longer shrinks P while the remaining $(n+1)/2$ vertices are examined. The matching obtained by the algorithm is at most $n - (|U| + |R|) = (n+1)/2$ because it does not match any vertices in $U \cup R$.

However, there exists a perfect matching. We construct it by looking back at the game between the algorithm and adversary. We match the first $(n-1)/2$ vertices processed by the algorithm to vertices in $U \cup R$. For each step in which the algorithm rejected a vertex v, there was some vertex r added to R: we match v to r. For each step in which the algorithm matched a vertex v, it was matched to a vertex m and there was a u that was added to U. We match v to u. Thus, after $(n-1)/2$ vertices, we have matched all of U and R. We are left with vertices in $OFF \setminus (U \cup R)$, which can be matched in any way to the $(n+1)/2$ remaining vertices in ON because that is precisely their set of neighbours.

For an example, take $n = 7$. Then the degrees have to be 4 and the adversary constructs M, U, and R when the first 3 online vertices are being examined. An example where the algorithm matches the first online vertex, rejects the second, and matches the third is shown in Figure 1. The

online vertices examined after an offline vertex is added to M will be neighbours of this vertex. The online vertices examined after an offline vertex is added to $U \cup R$ will not be neighbours of this vertex. A perfect matching for this example is shown in Figure 2.

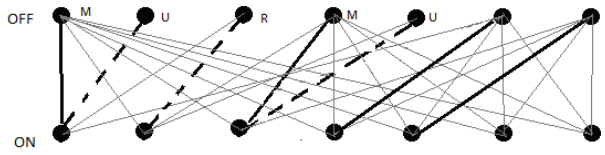


Figure 1: Offline vertices are labelled whenever they belong to M, U, or R. Dark lines correspond to matches made by the algorithm. Dotted lines correspond to edges such that, when the online vertex was processed, the adversary added the offline vertex to $U \cup R$. Light lines are all other edges. The algorithm obtains a matching of size 4.

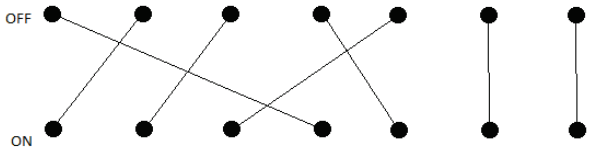


Figure 2: Same instance as before, now showing only the edges corresponding to a perfect matching obtained by the adversary.

□

5.2.2 Randomized priority algorithms

The randomized Ranking algorithm in the online model achieves approximation $1 - 1/e$ which is optimal for randomized online algorithms. In the ROM model, Ranking achieves an approximation of at least .696 and at most .727. Clearly, Ranking can be implemented as a randomized priority algorithm and this leaves open the question as to whether any fully randomized algorithm can improve upon the Ranking algorithm. In this section we are only able to derive a weak $\frac{53}{54}$ inapproximation result for bipartite matching with respect to the fully randomized priority model.

In comparison, we should mention the inapproximation results by Besser and Poloczek [8, 9] and Goel and Tripathi [30]. Both of these papers consider classes of randomized greedy algorithms for matching in general (not bipartite) graphs. We note again that approximations and inapproximations for general graphs do not necessarily carry over to the one-sided bipartite model. In the fully randomized priority model, Besser and Poloczek establish a $\frac{134}{135}$ inapproximation[3] for general graphs. Goel and Tripathi consider greedy algorithms for matching in general graphs with respect to two models, both of which can be implemented within the fully randomized priority model. With respect to their "query-commit" model, they establish an inapproxmation of .7916 in this model as well as a .75 inapproximation in their *vertex-iterative* model. Both of the Goel and Tripathi

[3]Besser and Polozek initially stated a $\frac{5}{6}$ inapproximation but then later in [9] clarified that claim to indicate that it only holds for a more restricted "degree-based model".

models query vertices to look for a an available match and once one is found, it must commit to that match.

The following argument emphasizes the difficulty in proving inapproximation results for (unrestricted) randomized priority algorithms. Using the Yao minimax lemma, it is sufficient to define a distribution on inputs such that every deterministic algorithm will suffer an inapproximation (in expectation) for that distribution. Consider the graph in Figure 3 and consider the following reasoning. Given that all online nodes have degree 2, it would seem that if v is the first node being seen (in the random ordering of the online vertices), then it will fail to choose u as its match with probability $\frac{1}{2}$ and hence the probability that the algorithm makes a mistake for the first online vertex it sees is $\frac{1}{6} > \frac{1}{9}$. This would yield the weak but improved bound of $\frac{17}{18}$. This reasoning is correct if the algorithm can only make decisions for an online node based on the degree of the node (and any previously seen information). For the first online node, there is no previous information, so that a *degree-based* algorithm would indeed achieve the improved $\frac{17}{18}$ approximation.

However, there is a way to order the vertices so that the probability of making a mistake for the first online vertex is less than $\frac{1}{6}$. Let a data item be described by $l_1 : (l_2, l_3)$, meaning that this corresponds to the data item of the online vertex with label l_1, having the offline vertices labelled with l_2 and l_3 as neighbours. Let the priority ordering given by the algorithm begin as follows: $v_1 : (u_1, u_2), v_1 : (u_1, u_3), v_2 : (u_1, u_2), v_2 : (u_1, u_3), v_3 : (u_1, u_2), v_3 : (u_1, u_3)$. At least one of the data items must be in the instance, regardless of how the nodes are permuted. Let us assume that upon receiving its first data item, the algorithm matches the online vertex to u_1.

We now analyze the probability that v is matched, but not to u. If v is not the first online vertex received, it is easy to see that the algorithm can achieve a perfect matching. When u is labelled as u_1, then v will be the first online vertex received, but the algorithm will match v to u and achieve a perfect match. So the only case when the algorithm can make a mistake is when v's other neighbour, u', is labelled as u_1. Since u' has all online nodes as neighbours, the one labelled v_1 will be received first. To make a mistake, then, u' has to be labelled as u_1 and v has to be labelled as v_1. Thus, the probability that the algorithm makes a mistake is $\frac{1}{9}$, and the algorithm achieves an approximation ratio of $\frac{8}{9} + \frac{1}{9} \times \frac{2}{3} = \frac{26}{27}$. This algorithm shows why the above degree-based argument is incorrect: it was able to receive v with probability 4/9, and conditioned on v being the first vertex, v is matched to u' with probability 1/4.

We are able to show that randomized priority algorithms cannot achieve optimality

Theorem 8. *No randomized priority algorithm can achieve an approximation better than* $\frac{53}{54}$.

Proof. Using the names as above, suppose the algorithm first considers $v_i : (v_j, v_k)$ and without loss of generality suppose the algorithm will match v_i to v_j. Then with probability $\frac{1}{3}$, v is named v_1, and with probability $\frac{1}{6}$, (u', u) is named (u_j, u_k) where u' is the other neighbor of v. Hence, any deterministic algorithm will fail to match v to u with probability 1/18 so that the expected number of matches is at most $\frac{1}{18} \cdot 2 + \frac{17}{18} \cdot 3 = 53/18$ while there is a perfect matching of size 3. □

5.3 Random order model for bipartite matching

Ranking is an online randomized algorithm, but as observed in [39], it also has a well-known interpretation as a ROM algorithm. It is equivalent to the Fixed Ranking algorithm in ROM: performing online Ranking on a graph $G = (V_1, V_2, E)$ (where V_1 is the online side and V_2 the offline side) is the same as deterministically matching to the first available neighbour in the graph

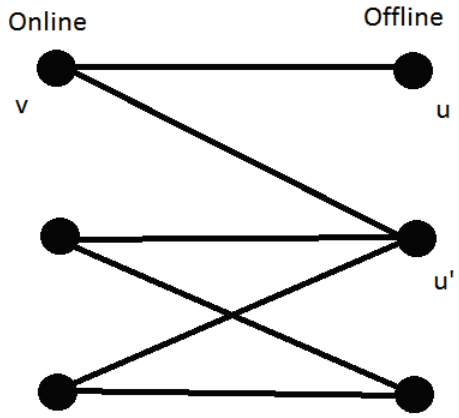


Figure 3: A graph showing the difficulty of proving priority randomized inapproximations.

$G' = (V_2, V_1, E)$, in the ROM model (now V_2 are the online vertices, and the ordering of V_1 used to decide which vertex is "first" corresponds to the online order in G). This means that deterministic ROM algorithms can achieve an approximation ratio of $1 - 1/e$.

It is known that algorithm Random does not get an approximation ratio better than $1/2$ in the online setting [39]. Figure 4 shows the hard instance used for $n = 6$. The online vertices are columns, and they arrive from right to left. The offline vertices are rows, and a 0 entry means there is no edge between the corresponding online and offline vertices, while a 1 means that there is. This is generalized in an obvious way to an instance of $2k$ vertices. The first k online vertices will have degree $k + 1$. They will have k common neighbours (corresponding to the first k rows in the matrix), plus an additional neighbour that has degree 1 (hence, the correct choice is to match the online vertex to this additional neighbour). The last k online vertices will have degree 1, each being matched to one of the first k rows.

Figure 4: Graph that is hard for algorithm Random.

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Random has trouble with this instance because it does not use information from previous online vertices when it has to "guess" the correct match of a given online vertex. In contrast, by using a random permutation of the offline vertices, Ranking tends to be biased in favor of vertices that have occurred less in the past. Suppose that several online vertices have been matched. In the permutation used by the algorithm, the remaining offline vertices from among the first k must appear after all the offline vertices that have already been matched. On the other hand, we know

nothing about the relative order among the first k offline vertices of the offline vertex u of degree 1 that is neighbour of the next online vertex. Thus, there are more permutations consistent with the choices made so far where u comes first among those available, so u is more likely to be matched. In contrast, in Random, at any point in time, all available vertices are equally likely to be matched.

Ranking has been well studied in ROM. It is interesting to note, however, that this is not the case for algorithm Random. Of course, it is very plausible that Random will not beat Ranking in ROM. However, it would be nice to know if it gets an approximation ratio greater than $1/2$ in this model. In fact, we do not even know whether Ranking is asymptotically better than Random. It could even be the case that Random is asymptotically better than Ranking. In the analysis of the following theorem we show that the performance of Random in ROM for the instance described above is worse than that of Ranking.

Theorem 9. *Algorithm Random achieves an asymptotic approximation ratio of at most* $3/4$ *in ROM.*

Proof. Consider the hard graphs of Figure 4. Fix a permutation of online vertices. Ranking will randomly choose a permutation of the offline vertices σ. Notice that Ranking and Random behave the same on online vertices of degree 1: they will match to its neighbour if it is still available. In particular, the behaviour of Ranking on degree 1 vertices reveals nothing about σ. We now see what happens to the vertices of degree $k+1$. For the first of these, both algorithms behave the same: the probability of picking any at random is equal to the probability of any being first (among the options) in the ranking.

Now, consider any online vertex u of degree $k+1$ and fix the choices of matches of previous vertices. We analyse the behaviour of Ranking and Random under the assumption that these matches were made previously by both algorithms. Let A be the set of offline vertices of degree $k+1$ that were previously matched to online vertices of degree $k+1$: the mistakes from the past. Let B be the set of offline vertices of degree $k+1$ that are still available currently, when choosing the match of u. Notice that all vertices in B must have also been available before. Because Ranking chose matches with vertices in A, then all vertices in A go before all vertices in B in σ. However, consider the unique offline vertex v of degree 1 that is neighbour of u: the correct match of u. This vertex could appear anywhere in σ, since it has never been considered before. This means that the probability of this vertex being first in σ among the available vertices is greater than that of any vertex in B. So the probability that Ranking matches u to v is higher than the probability that it matches u to any vertex in B.

Thus, for any permutation of online vertices and any fixed choices of matching for previous vertices, the probability that Ranking matches the current online vertex of degree $k+1$ to its neighbour of degree 1 (its correct match) is greater than or equal to the probability that Random does so. The size of the matching is equal to k plus the number of correct matchings: any incorrect match means there will be an online degree 1 vertex that will be unmatched. It is proved in [38] that the asymptotic approximation ratio of Ranking on these graphs is at most $3/4$. Thus, this bound also holds for Random. □

We now turn our attention to deterministic ROM algorithms. The following shows that in ROM an inapproximation result using a specific small graph will not yield the same inapproximation for arbitrarily large graphs just by taking the union of disjoint copies of the small graph. Consider the bipartite graph with 2 vertices on each side, where one online vertex has degree 2 and the other has degree 1. By carefully choosing the neighbour of the degree 1 vertex, we get that no deterministic

algorithm can get an approximation ratio better than 0.75. Now consider a graph with 4 vertices, with 2 components, each with the form of the graph with 2 online vertices (see Figure 5).

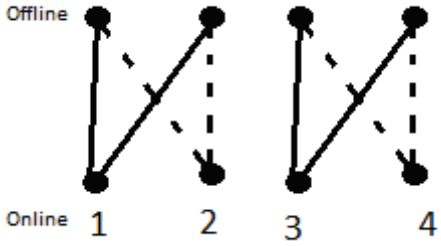


Figure 5: Vertices 2 and 4 have degree one, and the dotted lines show the possible neighbours.

Consider the following algorithm: match greedily at each step (so that if vertex 2 comes before 1 and 4 before 3, a perfect matching is obtained). Now, if the first vertex received (among 1,2,3,4) is vertex 1 or 3, match it to the leftmost neighbour. Otherwise, whenever there is a choice for vertex 1 because it occurred before vertex 2, choose the rightmost neighbour (analogously for vertex 3). We show that this algorithm achieves an approximation ratio of 0.875 on these graphs. If the graph consists of vertex 2 having the leftmost offline vertex as neighbour, the algorithm will get both vertices matched whenever vertex 1 occurs first in the permutation, and this happens with probability 1/4. The probability that vertex 2 comes before 1 is 1/2. Therefore the algorithm matches $1/2 * 2 + 1/4 * 2 + 1/4 * 1 = 1.75$ edges in expectation. If the graph consists of vertex 2 being matched to the option on the right, the algorithm will match both when 1 does not occur first, which happens with probability 3/4. Thus in this case the algorithm matches $3/4*2+1/4*1 = 1.75$ edges in expectation. The analysis for the other component can be done in the same way. That is, the performance of the algorithm on any such graph is $2 * 1.75 = 3.5$. Thus, the algorithm achieves an approximation ratio of $3.5/4 = 0.875$.

To the best of our knowledge, it is not known whether there is any deterministic ROM algorithm that achieves an asymptotic approximation ratio better than $1 - 1/e$. While we cannot answer this question, in our arXiv paper [47] we provide three deterministic algorithms that improve upon the $1 - 1/e$ bound for small (i.e., for $3 \leq n \leq 6$) graphs.

6 Conclusion and Open Problems

We considered the max-sat and bipartite matching problems in parallel, priority and ROM models. Our results were nformally outlined in section 1.1. Given the precise models being considered, we now restate those informal statements and elaborate on each of these results.

- For the max-sat problem, we show that the Buchbinder and Feldman [16] de-randomization method can be applied to obtain a deterministic parallel width $2n$ online $\frac{3}{4}$ approximation algorithm.

  This result is stated precisely in Theorem 1 as to the input model and the meaning of "width". Like Johnson's algorithm, this result holds for the max-sat input model 1 as defined in section

2.2.1. The width model is the width-k model as discussed in section 2.2.2. It is the online levelled tree width model that does not cut any branches.

- We then show (in a model more general than what is needed for the above $\frac{3}{4}$ approximation), that exponential online width is required to improve upon this ratio even for the exact max-2-sat problem for which Johnson's algorithm already achieves a $\frac{3}{4}$ approximation.

  Both the input and width model for this inapproximation are more general than what is needed for Theorem 1. Namely, Theorem 2 proves that the inapproximation result holds for input model 2 and for the tree width-cut model that also allows for cutting any branch. For the most general max sat input model 3, we can only show a $\frac{5}{6}$ inapproximation.

- For online bipartite matching we show that constant width (or even width $\frac{\log n}{\log \log n}$) cannot asymptotically beat the trivial $\frac{1}{2}$ approximation achieved by any greedy maximal matching algorithm. This implies that more than $\log \log n - \log \log \log n$ advice bits are needed to asymptotically improve upon the $\frac{1}{2}$ approximation achieved by any greedy maximal matching algorithm.

  For online bipartite matching we only consider the one-sided input model where nodes on one side are considered offline (i.e. their labels are known a-priori) and on the other side, the online nodes arrive sequentially, each online node described by its adjacent offline nodes. The width models require some care. Theorem 6 shows that no constant width k (without cut) algorithm can achieve an aysymptotic approximation better than $\frac{1}{2}$. However, the more precise $\frac{\log n}{\log \log n}$ bound (Corollary 1) was only estalished for the max-of-k model.

- For bipartite matching, we also show that the ability to sort the input items as in the priority model cannot compensate for the absence of randomization.

  More precisely, Theorem 5.2.1 shows that no adaptive priority algorithm can provide an approximation that is asymptotically better than the $\frac{1}{2}$ approximation achieved by any greedy algorithm.

- Finally, we make some observations about bipartite matching in the fully randomized priority model and in the ROM model.

  Section 5.2.2 provides a (seemingly) weak $\frac{53}{54}$ inapproximation for fully randomized priority algorithms exposing our current inability to properly understand this model. In contrast the currently best known approximation (.696) for fully randomized priority algorithms is achieved by the randomized Ranking algorithm in the ROM model.

  In Section 5.2.2 we show that an inapproximation in the ROM model established for a specific example does not extend to become an asymptotic result as it would in the adversarial online model. Ranking provides the currently best known *deterministic* approximation (namely, $1 - \frac{1}{e}$) in the ROM model. With regard to randomized algorithms in the ROM model, we do not know if Random (i.e. the algorithm that uniformly at random chooses an available offline node) achieves an approximation better than the $1 - \frac{1}{e}$ approximation of any fixed ordering of the offline nodes. Theorem 5.3 provides a .75 inapproximation for Random in the ROM model which is a weaker result than the inapproximation for Ranking in the ROM model.

Our results mainly apply to deterministic algorithms in variants of the online model. We call attention to some basic open problems. Namely,

- Approximation bounds for the equivalent models of max-of-k and $\log k$ advice (in the tape advice model) are non-uniform (i.e., for each input size n a new algorithm is needed) and require exponential time. To what extent can we make these results uniform and efficient? Does there exist a precise "realistic" model for advice that is reasonably general and analyzable?
- Is there a provable difference (in terms of achieveable approximation ratios) between the max-of-k and width k models (or between width k and width-cut k for max-sat or bipartite matching?
- Does there exist a deterministic online constant width algorithm for max-sat that can achieve approximation $\frac{3}{4}$?
- Is there a provable difference between input models 2 and 3 for max-sat?
- What is the best approximation that can be achieved by a constant (even width 2) priority algorithm for bipartite matching? That is, each branch of the computation tree is a fixed or adaptive order priority algorithm. As far as we know, there have been no results for bipartite matching in the pBT model of Alekhnovich et al [2].
- Turning to ROM or fully randomized priority algorithms, can we derive new proof methods to improve upon the known inapproximation results? In particular, we would like to know if Random is better or worse than the randomized Ranking algorithm in the ROM model.

Two (and multiple) pass algorithms provide another interesting extension of online (or priority) algorithms. The deterministic two-pass online algorithms for max-sat ([50]) and bipartite matching ([23, 14]) provably improve upon the known bounds for deterministic one pass algorithms against an adversarial input and there is evidence (e.g. [52]) that greedy and two-pass algorithms perform well "in practice".

Our general goal is to see to what extent "simple combinatorial algorithms" can achieve approximation ratios close to known hardness results and in doing so can come close to or even improve upon the best known offline algorithms. Our work here has led to the simple algorithms we proposed in [47]. Although conceptually simple, such algorithms constitute a significant challenge for their analysis. Part of our agenda is to better understand when and how one can de-randomize online algorithms.

7 Acknowledgements

We thank Matthias Poloczek for many helpful comments. This work was supported by NSERC Discovery Grant RGPIN-2017-06551

References

[1] G. Aggarwal, G. Goel, C. Karande, and A. Mehta, *Online vertex-weighted bipartite matching and single-bid budgeted allocations*, SODA, (2011), pp. 1253–1264.

[2] M. Alekhnovich, A. Borodin, J. Buresh-Oppenheim, R. Impagliazzo, A. Magen, and T. Pitassi, *Toward a model for backtracking and dynamic programming*, Computational Complexity, 20 (2011), pp. 679–740.

[3] S. Angelopoulos and A. Borodin, *Randomized priority algorithms*, Theoretical Computer Science, 411 (2010), pp. 2542–2558.

[4] J. Aronson, M. Dyer, A. Frieze, and S. Suen, *Randomized greedy matching II*, Random Structures and Algorithms, 6 (1995), pp. 55–73.

[5] A. Avidor, I. Berkovitch, and U. Zwick, *Improved approximation algorithms for max nae-sat and max sat*, WAOA, (2005), pp. 27–40.

[6] Y. Azar, I. Gamzu, and R. Roth, *Submodular max-sat*, ESA, (2011), pp. 323–334.

[7] B. Bahmani and M. Kapralov, *Improved bounds for stochastic online matching*, ESA, (2010), pp. 170–181.

[8] B. Besser and M. Poloczek, *Greedy matching: Guarantees and limitations*, Algorithmica, 77 (2017), pp. 201–234.

[9] ——, *Erratum to: Greedy matching: Guarantees and limitations*, Algorithmica, 80 (2018), pp. 410–413.

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

[11] H.-J. Böckenhauer, D. Komm, R. Královič, and R. Královič, *On the advice complexity of the k-server problem*, ICALP, (2011), pp. 207–218.

[12] A. Borodin, J. Boyar, K. S. Larsen, and N. Mirmohammadi, *Priority algorithms for graph optimization problems*, Theoretical Computer Science, 411 (2010), pp. 239–258.

[13] A. Borodin, M. N. Nielsen, and C. Rackoff, *(Incremental) priority algorithms*, Algorithmica, 37 (2003), pp. 295–326.

[14] A. Borodin, D. Pankratov, and A. Salehi-Abari, *On conceptually simple algorithms for variants of online bipartite matching*, in Approximation and Online Algorithms - 15th International Workshop, WAOA 2017, Vienna, Austria, September 7-8, 2017, Revised Selected Papers, 2017, pp. 253–268.

[15] J. Boyar, L. M. Favrholdt, C. Kudahl, K. S. Larsen, and J. W. Mikkelsen, *Online algorithms with advice: A survey*, ACM Comput. Surv., 50 (2017), pp. 19:1–19:34.

[16] N. Buchbinder and M. Feldman, *Deterministic algorithms for submodular maximization problems*, SODA, (2016), pp. 392–403.

[17] N. Buchbinder, M. Feldman, J. S. Naor, and R. Schwartz, *A tight linear time (1/2)-approximation for unconstrained submodular maximization*, FOCS, (2012), pp. 649–658.

[18] N. Buchbinder, M. Feldman, J. S. Naor, and R. Schwartz, *A tight linear time* (1/2)*-approximation for unconstrained submodular maximization*, SIAM J. Comput., 44 (2015), pp. 1384–1402.

[19] J. Chen, D. K. Friesen, and H. Zheng, *Tight bound on Johnson's algoritihm for maximum satisfiability*, Journal of Computer and System Sciences, 58 (1999), pp. 622–640.

[20] K. Costello, A. Shapira, and P. Tetali, *Randomized greedy: new variants of some classic approximation algorithms*, SODA, (2011), pp. 647–655.

[21] S. Davis and R. Impagliazzo, *Models of greedy algorithms for graph problems*, Algorithmica, 57 (2009), pp. 269–317.

[22] N. R. Devanur, K. Jain, and R. D. Kleinberg, *Randomized primal-dual analysis of ranking for online bipartite matching*, SODA, (2013), pp. 101–107.

[23] C. Dürr, C. Konrad, and M. P. Renault, *On the power of advice and randomization for online bipartite matching*, in 24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark, 2016, pp. 37:1–37:16.

[24] Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén, *Online computation with advice*, Theor. Comput. Sci., 412 (2011), pp. 2642–2656.

[25] L. Engebretsen, *Simplified tight analysis of Johnson's algorithm*, Information Processing Letters, 92 (2004), pp. 207–210.

[26] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, *On graph problems in a semi-streaming model*, Theor. Comput. Sci., 348 (2005), pp. 207–216.

[27] J. Feldman, A. Mehta, V. Mirrokni, and S. Muthukrishnan, *Online stochastic matching: Beating 1-1/e*, FOCS, (2009), pp. 117–126.

[28] A. Goel, M. Kapralov, and S. Khanna, *On the communication and streaming complexity of maximum bipartite matching*, in Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012, 2012, pp. 468–485.

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

[30] G. Goel and P. Tripathi, *Matching with our eyes closed*, in 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012, 2012, pp. 718–727.

[31] M. M. Halldórsson, K. Iwama, S. Miyazaki, and S. Taketomi, *Online independent sets*, COCOON, (2000), pp. 202–209.

[32] J. Håstad, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859.

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

[34] N. Huang and A. Borodin, *Bounds on double-sided myopic algorithms for unconstrained non-monotone submodular maximization*, ISAAC, (2014), pp. 528–539.

[35] K. Iwama and S. Taketomi, *Removable online knapsack problems*, ICALP, (2002), pp. 293–305.

[36] P. Jaillet and X. Lu, *Online stochastic matching: new algorithms with better bounds*, Math Operations Research, (2014), pp. 624–664.

[37] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. Syst. Sci., 9 (1974), pp. 256–278.

[38] C. Karande, A. Mehta, and P. Tripathi, *Online bipartite matching with unknown distributions*, STOC, (2011), pp. 587–596.

[39] R. M. Karp, U. V. Vazirani, and V. V. Vazirani, *An optimal algorithm for on-line bipartite matching*, STOC, (1990), pp. 352–358.

[40] T. Kesselheim, K. Radke, A. Tönnis, and B. Vöcking, *An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions*, ESA, (2013), pp. 589–600.

[41] A. Madry, *Navigating central path with electrical flows: From flows to matchings, and back*, in 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA, 2013, pp. 253–262.

[42] M. Mahdian and Q. Yan, *Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs*, STOC, (2011), pp. 597–606.

[43] V. H. Manshadi, S. O. Gharan, and A. Saberi, *Online stochastic matching: online actions based on offline statistics*, SODA, (2011), pp. 1285–1294.

[44] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani, *Adwords and generalized on-line matching*, FOCS, (2005), pp. 264–273.

[45] J. W. Mikkelsen, *Randomization can be as helpful as a glimpse of the future in online computation*, CoRR, abs/1511.05886 (2015).

[46] S. Muthukrishnan, *Data streams: Algorithms and applications*, Foundations and Trends in Theoretical Computer Science, 1 (2005).

[47] N. Pena and A. Borodin, *On the limitations of deterministic de-randomizations for online bipartite matching and max-sat*, CoRR, abs/1608.03182 (2016).

[48] M. Poloczek, *Bounds on greedy algorithms for MAX SAT*, ESA, (2011), pp. 37–48.

[49] M. Poloczek and G. Schnitger, *Randomized variants of Johnson's algorithm for MAX SAT*, SODA, (2011), pp. 656–663.

[50] M. Poloczek, G. Schnitger, D. P. Williamson, and A. van Zuylen, *Greedy algorithms for the maximum satisfiability problem: Simple algorithms and inapproximability bounds*, SIAM J. Comput., 46 (2017), pp. 1029–1061.

[51] M. Poloczek and D. P. Williamson, *An experimental evaluation of fast approximation algorithms for the maximum satisfiability problem*, in Experimental Algorithms - 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings, 2016, pp. 246–261.

[52] ——, *An experimental evaluation of fast approximation algorithms for the maximum satisfiability problem*, ACM Journal of Experimental Algorithmics, 22 (2017).

[53] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Commun. ACM, 28 (1985), pp. 202–208.

[54] A. van Zuylen, *Simpler 3/4-approximation algorithms for max sat*, WAOA, (2011), pp. 188–197.

[55] M. Yannakakis, *On the approximation of maximum satisfiability*, SODA, (1992), pp. 1–9.

[56] A. C. Yao, *New algorithms for bin packing*, J. ACM, 27 (1980), pp. 207–227.

[57] C. Yung, *Inapproximation result of exact max-2-sat. Unpublished.*