SIAM J. COMPUT. © 2017 Society for Industrial and Applied Mathematics
Vol. 46, No. 3, pp. 1029–1061

GREEDY ALGORITHMS FOR THE MAXIMUM SATISFIABILITY PROBLEM: SIMPLE ALGORITHMS AND INAPPROXIMABILITY BOUNDS*

MATTHIAS POLOCZEK†, GEORG SCHNITGER‡, DAVID P. WILLIAMSON†, AND ANKE VAN ZUYLEN§

Abstract. We give a simple, randomized greedy algorithm for the maximum satisfiability problem (MAX SAT) that obtains a $\frac{3}{4}$-approximation in expectation. In contrast to previously known $\frac{3}{4}$-approximation algorithms, our algorithm does not use flows or linear programming. Hence we provide a positive answer to a question posed by Williamson in 1998 on whether such an algorithm exists. Moreover, we show that Johnson's greedy algorithm cannot guarantee a $\frac{3}{4}$-approximation, even if the variables are processed in a random order. Thereby we partially solve a problem posed by Chen, Friesen, and Zheng in 1999. In order to explore the limitations of the greedy paradigm, we use the model of priority algorithms of Borodin, Nielsen, and Rackoff. Since our greedy algorithm works in an online scenario where the variables arrive with their set of undecided clauses, we wonder if a better approximation ratio can be obtained by further fine-tuning its random decisions. For a particular information model we show that no priority algorithm can approximate Online MAX SAT within $\frac{3}{4} + \varepsilon$ (for any $\varepsilon > 0$). We further investigate the strength of deterministic greedy algorithms that may choose the variable ordering. Here we show that no adaptive priority algorithm can achieve approximation ratio $\frac{3}{4}$. We propose two ways in which this inapproximability result can be bypassed. First we show that if our greedy algorithm is additionally given the variable assignments of an optimal solution to the canonical LP relaxation, then we can derandomize its decisions while preserving the overall approximation guarantee. Second we give a simple, deterministic algorithm that performs an additional pass over the input. We show that this 2-pass algorithm satisfies clauses with a total weight of at least $\frac{3}{4}\mathrm{OPT}_{LP}$, where OPT_{LP} is the objective value of the canonical linear program. Moreover, we demonstrate that our analysis is tight and detail how each pass can be implemented in linear time.

Key words. approximation algorithms, greedy algorithms, maximum satisfiability problem, priority algorithms, randomized algorithms

AMS subject classifications. 68W25, 68W20, 68W40, 68R01

DOI. 10.1137/15M1053369

1. Introduction.

Greedy algorithms are one of the most fundamental algorithmic concepts and typically among the first things one tries when facing a new problem. Their striking advantages are their simplicity and efficiency. These features make greedy algorithms preferable in practice, even if they may not achieve the performance

*Received by the editors December 21, 2015; accepted for publication (in revised form) February 3, 2017; published electronically June 20, 2017. Preliminary versions were presented at the ACM-SIAM Symposium on Discrete Algorithms (SODA11), the European Symposium on Algorithms (ESA11), the Workshop on Approximation and Online Algorithms (WAOA11), and the Latin American Theoretical INformatics Symposium (LATIN14).

http://www.siam.org/journals/sicomp/46-3/M105366.html

Funding: The first author was supported in part by the Alexander von Humboldt Foundation within the Feodor Lynen program and by NSF grant CCF-1115256. The second author was supported in part by DFG grant DFG SCHN 503/6-1. The third author was supported in part by NSF grant CCF-1115256. The fourth author was supported in part by a Suzann Wilson Matthews Summer Research Award.

†School of Operations Research and Information Engineering, Cornell University, Ithaca, NY 14853 (poloczek@cornell.edu, dpw@cs.cornell.edu).

‡Institute of Computer Science, Goethe University Frankfurt am Main, Frankfurt, Germany (georg@thi.cs.uni-frankfurt.de).

§Department of Mathematics, College of William and Mary, Williamsburg, VA 23185 (anke@wm.edu).

of more sophisticated techniques. However, sometimes they do.

In this article we study greedy algorithms for the maximum satisfiability problem (MAX SAT). MAX SAT is a central problem in discrete optimization: We are given n Boolean variables $x_1, \ldots, x_n$ and m clauses that are conjunctions of the variables or their negations. Each clause C_j has an associated weight $w_j \geq 0$. We say a clause is *satisfied* if one of its positive variables is set to true (equivalently, to one) or if one of its negated variables is set to false (resp., zero). The goal is to find an assignment of truth values to the variables so as to maximize the total weight of the satisfied clauses. The problem is NP-hard via a trivial reduction from satisfiability. Of particular interest are the following MAX SAT versions: in MAX EkSAT every clause has length *exactly k*, and in MAX kSAT every clause has length *at most k*.

Thus, we settle for algorithms that are guaranteed to produce a solution which is within a certain factor of the optimum. Formally, we say we have an α-approximation algorithm for MAX SAT if we have a polynomial-time algorithm that computes an assignment whose total weight of satisfied clauses is at least α times that of an optimal solution, and this performance guarantee holds for every input; then we call α the *approximation ratio* of the algorithm. A randomized α-approximation algorithm is a randomized polynomial-time algorithm such that the expected weight of the satisfied clauses is at least α times that of an optimal solution.

We briefly review the related work on MAX SAT and priority algorithms before we state our contributions in section 1.2.

1.1. Related work.

MAX SAT. The 1974 paper of Johnson [22], which introduced the notion of an approximation algorithm, also gave a $\frac{1}{2}$-approximation algorithm for MAX SAT. This greedy algorithm was later shown to achieve a $\frac{2}{3}$-approximation by Chen, Friesen, and Zheng [13] (see also the streamlined analysis of Engebretsen [16]). Poloczek [27] simplified the algorithm and showed that it even obtains $\frac{2}{3}$ of the optimum of the canonical linear programming (LP) relaxation.

Yannakakis [34] gave the first $\frac{3}{4}$-approximation algorithm for MAX SAT; it uses network flow computation and linear programming as subroutines. Goemans and Williamson [18] showed how to use randomized rounding of a linear program to obtain a $\frac{3}{4}$-approximation algorithm for MAX SAT. Recently, Chan et al. [12] showed that any polynomial-sized linear program for Max Cut has an integrality gap of $\frac{1}{2}$. MAX CUT is easily seen to be a special case of MAX SAT; in particular, a standard reduction shows that the existence of a polynomial-sized LP for MAX 2SAT with an integrality gap larger than $\frac{3}{4}$ would also give such a "small" LP with integrality gap greater than $\frac{1}{2}$ for MAX CUT.

Approximation algorithms based on semidefinite programming have led to even better performance guarantees. In a seminal paper Goemans and Williamson [19] used semidefinite programming for MAX 2SAT. Concluding a series of papers [17, 26], the (currently) best algorithm for MAX 2SAT, due to Lewin, Livnat, and Zwick [25], achieves a 0.94-approximation. Austrin [4] showed that no polynomial-time algorithm can achieve a better approximation ratio assuming the Unique Games Conjecture (UGC) (see, for instance, section 16.5 in the book of Williamson and Shmoys [33] for a survey of the UGC). Håstad [20] obtained a slightly worse inapproximability bound under the assumption P $\neq$ NP: he showed that no polynomial-time algorithm can approximate MAX 2SAT within $\frac{21}{22} + \varepsilon$ for any $\varepsilon > 0$ unless P = NP holds. Karloff and Zwick [23] gave a $\frac{7}{8}$-approximation algorithm for satisfiable MAX 3SAT instances. Zwick [37] showed via a computer assisted analysis that their algorithm also achieves this guarantee for unsatisfiable MAX 3SAT instances. This is the best possible approximation ratio, since Håstad [20] showed that it is NP-hard to approximate MAX E3SAT within $\frac{7}{8}+\varepsilon$ for any $\varepsilon > 0$. The currently best approximation ratio of 0.797 for MAX SAT is achieved by a "hybrid" algorithm due to Avidor, Berkovitch, and Zwick [5] that runs several algorithms based on semidefinite programming and outputs the best assignment.

Priority algorithms. The algorithmic results we present in this article are contrasted with an investigation of the limitations of greedy algorithms for MAX SAT. In order to formalize the intuitive understanding of what comprises a greedy algorithm, we utilize the framework of *priority algorithms* devised by Borodin, Nielsen, and Rackoff [9].

The framework formalizes the idea that greedy algorithms explore the input myopically and make an irrevocable decision for each piece (see also the "informal definition" of greedy algorithms by Kleinberg and Tardos [24, p. 115]). The crucial idea of Borodin, Nielsen, and Rackoff [9] is to regard the input of the algorithm as built from *data items*: Each data item encapsulates a small part of the input. In the case of MAX SAT our data item consists of a variable, say x, and the set of *undecided clauses* that x appears in. We say a clause of x is undecided if it has not been satisfied by previous assignments to other variables.

The framework was extended to graph problems by Davis and Impagliazzo [15] as well as Borodin et al. [8]; they had to overcome problems arising from the fact that different data items are related in the context of graph problems. In particular, nodes may appear as neighbors in the data items of other nodes, and one must ensure that the whole set of data items forms a valid instance. Randomized priority algorithms were introduced by Angelopoulos and Borodin [2].

It is important to note that the model does not impose any restrictions on the resources, like time or space, that the priority algorithm uses. Angelopoulos and Borodin [2] state that any bound utilizes "*the nature and the inherent limitations of the algorithmic paradigm, rather than resource constraints.*"

1.2. Our contributions.

This article is devoted to simple approximation algorithms for MAX SAT. The starting point for our investigations is the following research problem posed by Williamson in his lecture notes [32, p. 45]:

Can you get a $\frac{3}{4}$-approximation algorithm for MAX SAT without solving an LP?

We begin with a summary of our contributions.

Johnson's algorithm with random variable order. Johnson's algorithm for MAX SAT [22] considers the variables in an arbitrary ordering: For each variable x_i it considers the set of not yet satisfied clauses that x_i appears in and makes a deterministic decision where shorter clauses receive a higher priority. Chen, Friesen, and Zheng [13] proved that Johnson's algorithm achieves a $\frac{2}{3}$-approximation and posed the problem of analyzing the variant of Johnson's algorithm that processes the variables in a *random* order.

In this context it is noteworthy that Poloczek and Schnitger [28] showed that any 2CNF formula has an *optimal variable order*, i.e., there always exists an ordering of the variables such that Johnson's algorithm obtains an *optimal* assignment. Since MAX 2SAT is APX-hard, we cannot hope to find such an ordering efficiently (unless P = NP).

In section 2 we present a family of 2CNF formulae such that Johnson's algorithm performs worse than $\frac{3}{4}$ with high probability, even if the variable ordering is chosen at random. In particular, our family of instances has a vanishing number of optimal orders.

Independently of the first presentation of this result by Poloczek and Schnitger [28], Costello, Shapira, and Tetali [14] showed that this randomized variant of Johnson's algorithm achieves an expected approximation ratio of at least $\frac{2}{3} + 0.0036$. Thus, choosing a random variable order indeed improves the performance guarantee of Johnson's algorithm, but the algorithm still performs worse than the methods of Yannakakis [34] and Goemans and Williamson [18] using linear programming.

A simple randomized $\frac{3}{4}$-approximation algorithm. Poloczek and Schnitger [28] provided a positive answer to Williamson's question whether it is possible to obtain a $\frac{3}{4}$-approximation algorithm for MAX SAT without solving a linear program. They gave a randomized algorithm with the following particularly simple structure: Given a fixed ordering of the variables, the algorithm makes a random assignment to each variable in sequence, in which the probability of assigning each variable true or false depends on the current set of undecided clauses the variable appears in.

Subsequently, both the analysis and the algorithm itself were simplified, thereby yielding several variants that rely on the same principles and have the same structure (in particular, all work in the online scenario) but differ in the choice of assignment probabilities. Poloczek [27] obtained the stronger performance guarantee of $\frac{3}{4}\mathrm{OPT}_{LP}$, where OPT_{LP} is the value of the canonical LP relaxation, and a shorter analysis. Van Zuylen [31] gave elegant assignment probabilities and also provided a simpler analysis of the algorithm. In 2012 Buchbinder et al. [11, 10], as a special case of their work on maximizing submodular functions, gave a randomized $\frac{3}{4}$-approximation algorithm for MAX SAT with the same structure as these previous algorithms.

Poloczek, Williamson, and van Zuylen [30] showed that the assignment probabilities of [11] are the same as in [31]. Moreover, Poloczek, Williamson, and van Zuylen [30] gave a simple interpretation of the greedy algorithm using the approach of [11], which we present in detail in section 3. In this article we combine it with the analysis of Poloczek [27] to prove the stronger guarantee of $\frac{3}{4}\mathrm{OPT}_{LP}$. In Appendix A we describe how to implement the algorithm in linear time.

Priority algorithms for MAX SAT. Our algorithmic results motivate an investigation of the limitations of the greedy paradigm for MAX SAT. To this end, we utilize the framework of priority algorithms, where the input is considered to be a collection of data items. Our data items for MAX SAT contain the name of a variable, say x, to be processed and a list of undecided clauses x appears in. For each such clause c the following information is revealed:

(i) the sign of variable x in c,

(ii) the weight of c,

(iii) a list of the names of still unfixed variables appearing in c. No sign is revealed.

(Identical clauses are merged by adding their weights.)

Thus, we restrict access to information that, at least at first sight, is helpful only in revealing the global structure of the formula.

As we pointed out above, the variable order is an important parameter for greedy algorithms. Thus, we consider *adaptive* priority algorithms, a class of deterministic algorithms that may submit a new ordering on the set of data items in each step, without looking at the actual input. Then the algorithm receives the smallest data item according to the ordering and has to make an *instant* and *irrevocable* assignment to the respective variable given in the data item. By an “instant” decision we mean that the algorithm must fix the variable before it receives the next data item. Recall that the priority model does not impose any restrictions on the running time of the algorithm.

Johnson’s algorithm [22] and our randomized greedy algorithm (as well as its other variants) can be implemented as a priority algorithm with our data type, even assuming a worst-case ordering. However, no deterministic greedy algorithm with approximation ratio $\frac{3}{4}$ is known. For this data type, is randomization more powerful than determinism?

We show in section 5 that adaptive priority algorithms cannot achieve approximation ratio $\frac{3}{4}$. Thus, even allowing a priority algorithm to choose the next variable to set cannot result in a $\frac{3}{4}$-approximation algorithm, suggesting that randomization is indeed necessary to achieve a simple $\frac{3}{4}$-approximation algorithm.

An interesting aspect of our randomized greedy algorithm is that it does not utilize an intricate ordering of the variables. Quite the contrary, carefully adjusted assignment probabilities alone are essential for this algorithm—is it possible that the approximation ratio can be improved by some elaborate fine-tuning of its probabilities?

Therefore, we study Online MAX SAT, where the adversary reveals data items successively, and the algorithm must fix the corresponding variable instantly and irrevocably. In section 4 we show that no randomized priority algorithm achieves approximation ratio $\frac{3}{4} + \varepsilon$ for Online MAX SAT; therefore our randomized algorithm is optimal under these conditions.

A deterministic LP rounding scheme. One possibility to bypass our inapproximability result for deterministic greedy algorithms is to reveal more information to the algorithm. Indeed, if the randomized greedy algorithm additionally receives for each variable the assignment according to an optimal solution to the canonical LP relaxation, then the decisions can be made deterministically while preserving the overall approximation guarantee. This observation was made first by van Zuylen [31]; in section 6 we present a simpler rounding scheme that was given by Poloczek, Williamson, and van Zuylen [30].

Note that this approach requires solving a linear program; hence it does not yield a linear time algorithm.

A deterministic 2-pass algorithm. Our randomized greedy algorithm runs in linear time and yields a $\frac{3}{4}$-approximation in expectation. Can we compute a Boolean assignment which *guarantees* the same approximation ratio using only linear time?

In section 7 we present a deterministic $\frac{3}{4}$-approximation algorithm that performs two passes over the input. The first pass is essentially the same as the randomized greedy algorithm: the variables are considered in an arbitrary ordering, and for each variable x_i the algorithm computes an assignment probability; the crucial difference is that no variable is fixed yet.

At the end of the first pass the algorithm has computed an assignment probability σ_i for each variable x_i, such that fixing each x_i to true *independently* with probability σ_i would satisfy clauses with a total expected weight of at least $\frac{3}{4}\mathrm{OPT}_{LP}$. Instead of setting the variables randomly, our algorithm performs a second pass over the variables and derandomizes the random assignment via the method of conditional expectations. An important feature of this algorithm is that both passes can be implemented in linear time, as we describe in Appendix B.

Does the second pass improve the performance of the algorithm? In section 7.3

we present an instance for which the randomized greedy algorithm obtains only a $\frac{3}{4}$-approximation with high probability, but the deterministic 2-pass algorithm finds an optimal solution. Furthermore, we present a clause set that demonstrates the tightness of our worst-case analysis in section 7.4.

A very recent experimental evaluation by Poloczek and Williamson [29] shows that the 2-pass algorithm performs extremely well on a comprehensive set of benchmark instances from the SAT and the MAX SAT competitions [7, 3]. In particular, on the interesting set of instances stemming from industrial applications, the 2-pass algorithm satisfies more than 99% of the clauses on average, while being extremely fast. Also our randomized greedy algorithm and Johnson's algorithm perform far better than their worst-case guarantees let us expect: they satisfy more than 95% of the clauses on average.

1.3. The structure of the article. In section 2 we study Johnson's algorithm with a random variable order. Our randomized greedy algorithm that achieves a $\frac{3}{4}$-approximation is presented in section 3. In section 4 we show the tight inapproximability bound for online priority algorithms. We explore the limitations of adaptive priority algorithms and show that no deterministic greedy algorithm achieves a $\frac{3}{4}$-approximation in section 5. Section 6 extends the ideas of the randomized greedy algorithm to a deterministic LP-rounding scheme. In section 7 we present the simple, deterministic 2-pass algorithm that obtains a $\frac{3}{4}$-approximation in linear time.

2. Johnson's algorithm with random variable ordering.

We begin by studying the variant of Johnson's algorithm proposed by Chen, Friesen, and Zheng [13] that considers the variables $x_1, x_2, \ldots, x_n$ in a random order. Interestingly, using a random permutation instead of a worst-case ordering improves the approximation ratio by a small constant as was shown by Costello, Shapira, and Tetali [14].

We briefly summarize Johnson's algorithm: When deciding a variable x_i the algorithm computes the *modified weight* $\mu(x_i)$ ($\mu(\overline{x}_i)$) for both literals x_i and $\overline{x}_i$. Then the algorithm sets x_i to one if $\mu(x_i) \geq \mu(\overline{x}_i)$ and to zero otherwise.

We assume that literals that have been fixed to zero are removed from not yet satisfied clauses. In particular, the modified weight does not depend on removed literals in such clauses nor is such information used. Clauses with only one literal (left) are called *unit clauses*; note that they have only one more chance to be satisfied. In the case of a 2CNF formula the value of $\mu(x_i)$ (resp., of $\mu(\overline{x}_i)$) equals the sum of the weights of all undecided clauses that x_i (resp., $\overline{x}_i$) appears in, where the weights of unit clauses are doubled.

In what follows we give the following bound on the approximation ratio of this variant of Johnson's algorithm and therefore provide a partial answer to the question posed by Chen, Friesen, and Zheng [13] on what the approximation ratio of this variant is.

THEOREM 2.1. *The approximation ratio of Johnson's algorithm, when considering the variables in a random order, is at most*

$$2\sqrt{15} - 7 \approx 0.746 < \frac{3}{4}$$

with high probability as $n \to \infty$.

Proof. Our formula consists of variables $x_1, \ldots, x_n$, $y_1, \ldots, y_n$ and the following clauses:

(i) unit clauses $\overline{x}_i$ with (small) weight $S = \delta \cdot n$ and y_i with (large) weight

$L = (1-\varepsilon)\cdot n$ for $i = 1,\dots,n$,

(ii) all equivalences $x_i \leftrightarrow x_j$ with weight 1 for $1 \le i \ne j \le n$, and

(iii) all equivalences $x_i \leftrightarrow y_j$ with weight 1 for $1 \le i, j \le n$.

For the sake of simplicity we assume that S and L are integers. Note that an equivalence constraint $u \leftrightarrow v$ for two variables u, v is expressed as two 2-clauses $(\overline{u} \vee v), (u \vee \overline{v})$. In particular, if one of the variables is set by the algorithm, then the not yet satisfied 2-clause of the equivalence becomes a unit clause afterwards; i.e., if $u = 1$, then $(\overline{u} \vee v)$ becomes (v). For the sake of simplicity, we merge all unit clauses of each variable by adding up their weights. Note that the algorithm does not track whether a unit clause was present in the initial input or results from removed literals.

We choose δ to be a sufficiently small positive constant and carefully select ε to belong to the interval $\left[0, \frac{1}{2}\right]$. Thus, the weight S of the unit clause of each x-variable is indeed rather small compared with the weight L of the unit clauses that the y-variables appear in. Now consider the all-ones assignment. All equivalences and all y-units are satisfied and hence a total weight of

$$W = 2\cdot\binom{n}{2} + 2\cdot n^2 + (1-\varepsilon)\cdot n^2 = (4-\varepsilon)\cdot n^2 - n$$

is satisfied. Here the first summand corresponds to the equivalences between x-variables, the second stems from equivalences between x- and y-variables, and the third accounts for the unit clauses of the y-variables.

Now assume that π is a permutation of the variables that is chosen uniformly at random. How will Johnson's algorithm behave on π? The "first" x-variables that are processed will be set to zero and the "first" y-variables will be set to one.

When setting the ith x-variable x_j we expect that the number of x-variables set to zero is roughly the same as the number of y-variables that are fixed to one. Then the influences of the already fixed x- and y-variables on x_j almost cancel: The already fixed x-variables support $x_j = 0$ and the already fixed y-variables support $x_j = 1$. Any variable that has not been set yet does not influence the decision for x_j at all: the 2-clauses that it appears in, together with x_j, contain the variable x_j exactly once as positive and exactly once as negated literal. Thus, we expect the unit clause $\overline{x}_j$ with weight S to tip the scales and the algorithm to set $x_j = 0$.

To make this argument precise we consider a randomly permuted sequence of n x-variables that we assume will be set to zero and n y-variables that for now we assume all to be fixed to one. Let ξ_k for $1 \le k \le 2n$ be the event that in positions up to k the number of y-variables (resp., ones) deviates from the number of x-variables (zeros) by more than $S = \delta\cdot n$. The following lemma gives an upper bound on the probability of ξ_k.

LEMMA 2.2. *Let π be a random permutation of n x- and n y-variables. Then for any $k \in \{1,\dots,2n\}$, the probability that in the prefix of size k of π the number of x-variables deviates from the number of y-variables by more than δn is at most* $2e^{-\Theta(\delta^2 n)}$ *for any $\delta > 0$.*

Proof. Note that the probabilities of having a y-variable in some fixed position are not independent from each other. In particular, if there is a surplus of y-variables up to position $k-1$, then the conditional probability that the next variable in position k of the random permutation is a y-variable is less than $\frac{1}{2}$, since the whole permutation contains exactly n variables of each type. Nevertheless, we assume that each position up to k is a y-variable (resp., x-variable) with probability $\frac{1}{2}$ *independently* of any other position. By this, we only increase the (unconditional) probability of ξ_k. Then we apply the Chernoff–Hoeffding bound [21] to obtain that the probability that the numbers of y-variables and x-variables up to position k differ by more than S is at most $2e^{-\Theta(\delta^2 \cdot n)}$. $\square$

We observe that an x-variable is set to one only if for some prefix of the permutation the number of y-variables exceeds the number of x-variables by more than S. But then the event ξ_k must occur for some k; denote the probability of $\bigcup_k \xi_k$ by ρ. Applying the union bound for all $2n$ prefixes yields $\rho \leq 4n \cdot e^{-\Theta(\delta^2 \cdot n)}$. Hence all x-variables are set to zero with probability at least $1 - \rho$.

Let us shift our focus on the y-variables and consider the ith y-variable according to π, say y_k. Analogously to our above considerations for x_k, any variable that has not been set yet does not influence the decision for y_k. Assume that f_i x-variables are already fixed to zero. They push y_k toward zero, whereas the unit clause y_k of weight L pushes toward one: The difference $L - f_i$ decides. In particular, when f_i exceeds L eventually, the remaining y-variables will be fixed to zero. How many y-variables are set to one? We utilize that, for any time k, the number of x- and y-variables that have already been fixed differ by at most S with probability at least $1 - \rho$. Hence, the number of y-variables set to one belongs to the interval $[L - S, L + S]$ with probability at least $1 - \rho$.

Finally, we determine the expected score of Johnson's algorithm up to lower order terms. We may assume that all x-variables are set to zero and that the number of y-variables set to one belongs to the interval $[L - S, L + S]$. All equivalences between x-variables, all x-unit clauses, and all clauses $(\overline{x}_i \vee y_j)$ (with a combined weight of $2 \cdot \binom{n}{2} + n \cdot S + n^2 = 2 \cdot n^2 - n + n \cdot S$) are satisfied. Moreover, the combined weight of satisfied y-unit clauses and clauses $(\overline{y}_j \vee x_i)$ is at most $(L + S) \cdot L + (n - L + S) \cdot n$. Observe that the algorithm does not satisfy clauses with expected total weight of $L \cdot (n - L - S) + n \cdot (L - S)$, a loss which turns out to be larger than $\frac{W}{4}$, where W is the weight of the optimal assignment. Hence the expected satisfied weight is at most

$$2 \cdot n^2 + L^2 + (n - L) \cdot n + O(S \cdot n) = (2 + (1 - \varepsilon)^2 + \varepsilon) \cdot n^2 + O(\delta \cdot n^2).$$

We choose δ to be sufficiently small and hence the approximation ratio is at most

$$\alpha = \frac{2 + (1 - \varepsilon)^2 + \varepsilon}{4 - \varepsilon} = \frac{3 - \varepsilon + \varepsilon^2}{4 - \varepsilon}$$

as $n \to \infty$. A simple calculation shows that the quotient is minimized for $\varepsilon = 4 - \sqrt{15}$. Thus, the approximation ratio bounded above by

$$\begin{aligned} \alpha &= \frac{3 - 4 + \sqrt{15} + (16 - 8\sqrt{15} + 15)}{\sqrt{15}} \\ &= \frac{30 - 7\sqrt{15}}{\sqrt{15}} = 2\sqrt{15} - 7 \approx 0.746 \end{aligned}$$

with high probability. $\square$

Recall that every 2CNF formula has an optimal order; i.e., there is a variable order such that Johnson's algorithm produces an optimal assignment, as shown by Poloczek and Schnitger [28]. Our analysis, however, shows that picking a variable order with approximation ratio $\frac{3}{4}$ (or better) is extremely unlikely for this instance.

3. A simple randomized $\frac{3}{4}$-approximation algorithm.

In the following discussion we present a randomized greedy algorithm that achieves a $\frac{3}{4}$-approximation in expectation. We begin with a high level description of the algorithm: Consider greedy algorithms that set the variables x_i in sequence. A natural greedy algorithm sets x_i to true or false depending on which assignment increases the total weight of the satisfied clauses the most. An alternative to this algorithm would be to set each x_i so as to increase the total weight of the clauses that are not yet *unsatisfied* given the setting of the variable (a clause is unsatisfied if all the variables of the clause have been set and their assignment does not satisfy the clause).

Our algorithm is in a sense a randomized balancing of these two algorithms. It maintains a bound that is the average of two numbers, the total weight of the clauses satisfied so far and the total weight of the clauses that are not yet unsatisfied. For each variable x_i, it computes the amount by which the bound will increase if x_i is set true or false; one can show that the sum of these two quantities is always nonnegative. If one assignment causes the bound to decrease, the variable is given the other assignment (e.g., if assigning x_i true decreases the bound, then it is assigned false). Otherwise, the variable is set randomly with a bias towards the larger increase.

Now we give the formal description: We assume a fixed ordering of the variables, which for simplicity will be given as $x_1, x_2, \ldots, x_n$. As the algorithm proceeds, it will sequentially set the variables; let S_i denote the algorithm's setting of the first i variables.

Moreover, we denote by $W = \sum_{j=1}^m w_j$ the total weight of all the clauses. Let SAT_i be the total weight of clauses satisfied by S_i, and let UNSAT_i be the total weight of clauses that are unsatisfied by S_i—that is, clauses that have only variables from $x_1, \ldots, x_i$ and are not satisfied by S_i. Note that SAT_i is a lower bound on the total weight of clauses satisfied by our final assignment S_n (once we have set all the variables); furthermore, note that $W - \mathrm{UNSAT}_i$ is an upper bound on the total weight of clauses satisfied by our final assignment S_n. We let $B_i = \frac{1}{2}(\mathrm{SAT}_i + (W - \mathrm{UNSAT}_i))$ be the midpoint between these two bounds; we refer to it simply as the bound on our partial assignment S_i. For any assignment S to all of the variables, let $w(S)$ represent the total weight of the satisfied clauses. Then we observe that for the assignment S_n, $w(S_n) = \mathrm{SAT}_n = W - \mathrm{UNSAT}_n$, so that $w(S_n) = B_n$. Furthermore, $\mathrm{SAT}_0 = 0$ and $\mathrm{UNSAT}_0 = 0$, so that $B_0 = \frac{1}{2}W$.

Note that our algorithm will be randomized, so that S_i, SAT_i, UNSAT_i, and B_i are all random variables.

The goal of the algorithm is at each step to try to increase the bound; that is, we would like to set x_i randomly so as to increase $\mathbb{E}[B_i - B_{i-1}]$. We let t_i be the value of $B_i - B_{i-1}$ if we set x_i true, and f_i the value of $B_i - B_{i-1}$ if we set x_i false. Note that the expectation is conditioned on our previous setting of the variables $x_1, \ldots, x_{i-1}$, but we omit the conditioning for simplicity of notation. We will show momentarily that $t_i + f_i \geq 0$. Then the algorithm is as follows. If $f_i \leq 0$, we set x_i true; that is, if setting x_i false would not increase the bound, we set it true. Similarly, if $t_i \leq 0$ (setting x_i true would not increase the bound), we set x_i false. Otherwise, if either setting x_i true or false would increase the bound, we set x_i true with probability $\frac{t_i}{t_i+f_i}$.

LEMMA 3.1. *For $i = 1, \ldots, n$,*

$$t_i + f_i \geq 0.$$

Proof. We note that any clause of x_i that becomes unsatisfied by S_{i-1} and setting x_i true must then be satisfied by setting x_i false, and similarly any clause that becomes unsatisfied by S_{i-1} and setting x_i false must then be satisfied by setting x_i true. Let $\mathrm{SAT}_{i,t}$ be the clauses that are satisfied by setting x_i true given the partial assignment

S_{i-1}, and $\mathrm{SAT}_{i,f}$ be the clauses satisfied by setting x_i false given the partial assignment S_{i-1}. We define $\mathrm{UNSAT}_{i,t}$ ($\mathrm{UNSAT}_{i,f}$) to be the clauses unsatisfied by S_{i-1} and x_i set true (resp., false). Our observation above implies that $\mathrm{SAT}_{i,f} - \mathrm{SAT}_{i-1} \geq \mathrm{UNSAT}_{i,t} - \mathrm{UNSAT}_{i-1}$ and $\mathrm{SAT}_{i,t} - \mathrm{SAT}_{i-1} \geq \mathrm{UNSAT}_{i,f} - \mathrm{UNSAT}_{i-1}$.

Let $B_{i,t} = \frac{1}{2}(\mathrm{SAT}_{i,t} + (W - \mathrm{UNSAT}_{i,t}))$ and $B_{i,f} = \frac{1}{2}(\mathrm{SAT}_{i,f} + (W - \mathrm{UNSAT}_{i,f}))$. Then $t_i = B_{i,t} - B_{i-1}$ and $f_i = B_{i,f} - B_{i-1}$; our goal is to show that $t_i + f_i \geq 0$, or

$$\frac{1}{2}(\mathrm{SAT}_{i,t} + (W - \mathrm{UNSAT}_{i,t})) + \frac{1}{2}(\mathrm{SAT}_{i,f} + (W - \mathrm{UNSAT}_{i,f})) \\ - \mathrm{SAT}_{i-1} - (W - \mathrm{UNSAT}_{i-1}) \geq 0.$$

Rewriting, we want to show that

$$\frac{1}{2}(\mathrm{SAT}_{i,t} - \mathrm{SAT}_{i-1}) + \frac{1}{2}(\mathrm{SAT}_{i,f} - \mathrm{SAT}_{i-1}) \\ \geq \frac{1}{2}(\mathrm{UNSAT}_{i,f} - \mathrm{UNSAT}_{i-1}) + \frac{1}{2}(\mathrm{UNSAT}_{i,t} - \mathrm{UNSAT}_{i-1}),$$

and this follows from the inequalities of the previous paragraph. $\square$

The reference assignment. We will compare the algorithm to the value of an optimal solution to the standard LP relaxation of MAX SAT. Let us begin by describing the LP relaxation: The underlying binary program has decision variables $y_i \in \{0,1\}$, where $y_i = 1$ corresponds to x_i being set true for $1 \leq i \leq n$, and $z_j \in \{0,1\}$, where $z_j = 1$ with $1 \leq j \leq m$ corresponds to clause C_j being satisfied. Let P_j be the set of variables that occur positively in clause C_j and N_j be the set of variables that occur negatively. Then the LP relaxation is

$$\begin{array}{lll} \text{maximize} & \displaystyle\sum_{j=1}^{m} w_j z_j & \\ \text{subject to} & \displaystyle\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j & \forall C_j = \displaystyle\bigvee_{i \in P_j} x_i \vee \bigvee_{i \in N_j} \overline{x}_i, \\ & 0 \leq y_i \leq 1, & i = 1, \ldots, n, \\ & 0 \leq z_j \leq 1, & j = 1, \ldots, m. \end{array}$$

The value of a fractional assignment $y \in [0,1]^n$ is given by

$$w(y) = \sum_{j=1}^{m} w_j \cdot \min\left\{1; \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i)\right\}.$$

Note that the value of a fractional assignment can be larger by a factor of $\frac{4}{3}$ than that of the best Boolean one: an example are the four 2-clauses on two variables. In particular, if a clause set contains no unit clauses, i.e., no clause with only one literal, then the value of a best fractional solution equals the total clause weight.

In what follows, let y^* be an optimal assignment to the y-variables of the LP and denote by $\mathrm{OPT}_{\mathrm{LP}}$ its LP-value. Moreover, given a partial Boolean assignment S_i, let OPT_i be the assignment in which variables $x_1, \ldots, x_i$ are set as in S_i and $x_{i+1}, \ldots, x_n$ are set as in y^*. Then we have $w(\mathrm{OPT}_0) = \mathrm{OPT}_{LP}$ and $w(\mathrm{OPT}_n) = w(S_n)$. Note that we do not need the LP solution for the algorithm. We will use it only in the analysis.

The following lemma is at the heart of previous analyses (e.g., see section 2.2 in Poloczek and Schnitger [28] or Lemma III.1 of Buchbinder et al. [10]).

LEMMA 3.2. *For $i = 1, \dots, n$, the following holds:*

$$\mathbb{E}[w(\mathrm{OPT}_{i-1}) - w(\mathrm{OPT}_i)] \le \mathbb{E}[B_i - B_{i-1}].$$

Before we prove the lemma, we show that it leads straightforwardly to the desired approximation bound.

THEOREM 3.3. *Let $\mathrm{OPT}_{\mathrm{LP}}$ denote the optimal objective value of the standard LP relaxation and let W be the total weight of all clauses. Then the total weight satisfied in expectation by the randomized greedy algorithm is at least*

$$\mathbb{E}\left[w(S_n)\right] \ge \frac{2\mathrm{OPT}_{\mathrm{LP}} + W}{4} \ge \frac{3}{4}\mathrm{OPT}_{\mathrm{LP}}.$$

Proof. We sum together the inequalities from the lemma, so that

$$\sum_{i=1}^{n} \mathbb{E}[w(\mathrm{OPT}_{i-1}) - w(\mathrm{OPT}_i)] \le \sum_{i=1}^{n} \mathbb{E}[B_i - B_{i-1}].$$

Using the linearity of expectation and telescoping the sums, we get

$$\mathbb{E}[w(\mathrm{OPT}_0) - w(\mathrm{OPT}_n)] \le \mathbb{E}[B_n] - \mathbb{E}[B_0].$$

Thus, we have

$$\begin{aligned} \mathrm{OPT}_{LP} - \mathbb{E}[w(S_n)] &\le \mathbb{E}[w(S_n)] - \frac{1}{2}W \\ \mathrm{OPT}_{LP} + \frac{1}{2}W &\le 2\mathbb{E}[w(S_n)] \\ \frac{3}{4}\mathrm{OPT}_{LP} \le \frac{1}{2}\mathrm{OPT}_{LP} + \frac{1}{4}W &\le \mathbb{E}[w(S_n)] \end{aligned}$$

as desired, since $\mathrm{OPT}_{LP} \le W$. ∎

The following lemma is the key insight of proving the main lemma, and here we use the randomized balancing of the two greedy algorithms mentioned in the introduction.

LEMMA 3.4. *Let $t_i + f_i > 0$. Then*

$$\mathbb{E}[w(\mathrm{OPT}_{i-1}) - w(\mathrm{OPT}_i)] \le \max\left\{0, \frac{2t_i f_i}{t_i + f_i}\right\}.$$

Proof. Recall that $y_i^* \in [0,1]$ is the optimal fractional LP value for variable x_i. Assume that the algorithm fixes x_i to one. Then the difference $w(\mathrm{OPT}_{i-1}) - w(\mathrm{OPT}_i)$ is equal to the change in the value of the assignment if x_i is reduced from 1 in OPT_i to y_i^* in OPT_{i-1}. Any clause that contributes a positive amount to the difference $w(\mathrm{OPT}_{i-1}) - w(\mathrm{OPT}_i)$ must contain the negative literal $\overline{x}_i$, and the contribution of such a clause is increased by at most $(1 - y_i^*)$-times its weight. Thus, the total increase is upper-bounded by $(1 - y_i^*) \cdot (\mathrm{SAT}_{i,f} - \mathrm{SAT}_{i-1})$, where $\mathrm{SAT}_{i,f} - \mathrm{SAT}_{i-1}$ equals the total weight of clauses that would be newly satisfied by setting x_i to zero; recall that these are the clauses that are not satisfied by the algorithm's assignments to $x_1, \dots, x_{i-1}$ and contain the literal $\overline{x}_i$.

On the other hand, clauses that would become unsatisfied for $x_i = 0$ decrease the difference; recall that their total weight is $\mathrm{UNSAT}_{i,f} - \mathrm{UNSAT}_{i-1}$. These clauses must contain x_i positively and in particular contribute their full weight to $w(\mathrm{OPT}_i)$ but only a y_i^*-fraction to $w(\mathrm{OPT}_{i-1})$. Therefore, the total decrease related to these clauses is $(1-y_i^*) \cdot (\mathrm{UNSAT}_{i,f} - \mathrm{UNSAT}_{i-1})$. Combining both bounds gives that if x_i is fixed to one by the algorithm, then

$$\begin{aligned} & w(\mathrm{OPT}_{i-1}) - w(\mathrm{OPT}_i) \\ & \le (1-y_i^*) \cdot ((\mathrm{SAT}_{i,f} - \mathrm{SAT}_{i-1}) - (\mathrm{UNSAT}_{i,f} - \mathrm{UNSAT}_{i-1})) \\ & = (1-y_i^*) \cdot 2 \cdot f_i \end{aligned} \tag{3.1}$$

holds. Analogously, we obtain

$$w(\mathrm{OPT}_{i-1}) - w(\mathrm{OPT}_i) \le y_i^* \cdot 2 \cdot t_i \tag{3.2}$$

for the case that the algorithm fixes x_i to zero.

How does the algorithm fix the variable x_i? If $f_i \le 0$, then it fixes $x_i = 1$; hence $w(\mathrm{OPT}_{i-1}) - w(\mathrm{OPT}_i) \le 0$ by (3.1). The case $t_i \le 0$ is analogous. Thus, we assume that $t_i, f_i > 0$ and recall that then the algorithm fixes x_i to one with probability $\frac{t_i}{t_i+f_i}$ (and to zero otherwise). Therefore, we have

$$\begin{aligned} \mathbb{E}\left[w(\mathrm{OPT}_{i-1}) - w(\mathrm{OPT}_i)\right] &\le \frac{t_i}{t_i+f_i} \cdot (1-y_i^*) \cdot 2 \cdot f_i + \frac{f_i}{t_i+f_i} \cdot y_i^* \cdot 2 \cdot t_i \\ &= \frac{2t_i f_i}{t_i+f_i} \cdot (1 - y_i^* + y_i^*), \end{aligned}$$

which proves the lemma. ∎

Now we can prove the main lemma.

Proof of Lemma 3.2. First consider the case that $t_i = f_i = 0$ holds and recall that $t_i = B_{i,t} - B_{i-1}$, where we defined $B_{i-1} = \frac{1}{2}(\mathrm{SAT}_{i-1} + (W - \mathrm{UNSAT}_{i-1}))$ and $B_{i,t} = \frac{1}{2}(\mathrm{SAT}_{i,t} + (W - \mathrm{UNSAT}_{i,t}))$. Then $t_i = 0$ is equivalent to

$$\begin{aligned} & \mathrm{SAT}_{i,t} + (W - \mathrm{UNSAT}_{i,t}) - (\mathrm{SAT}_{i-1} + (W - \mathrm{UNSAT}_{i-1})) = 0, \\ & \mathrm{SAT}_{i,t} - \mathrm{SAT}_{i-1} = \mathrm{UNSAT}_{i,t} - \mathrm{UNSAT}_{i-1}, \end{aligned}$$

and analogously $f_i = 0$ gives $\mathrm{SAT}_{i,f} - \mathrm{SAT}_{i-1} = \mathrm{UNSAT}_{i,f} - \mathrm{UNSAT}_{i-1}$. Then $w(\mathrm{OPT}_{i-1}) - w(\mathrm{OPT}_i) = 0$ and $B_i = B_{i-1}$, regardless of the Boolean assignment to x_i.

If either $t_i \le 0$ or $f_i \le 0$, then the algorithm sets x_i deterministically so that the bound does not decrease: $B_i - B_{i-1} \ge 0$. Since then $t_i \cdot f_i \le 0$, we have by Lemma 3.4

$$\mathbb{E}[w(\mathrm{OPT}_{i-1}) - w(\mathrm{OPT}_i)] \le \max\{0, 2t_i f_i/(t_i+f_i)\} \le 0,$$

and the inequality holds.

If both $t_i, f_i > 0$, then

$$\begin{aligned} \mathbb{E}[B_i - B_{i-1}] &= \frac{t_i}{t_i+f_i}[B_{i,t} - B_{i-1}] + \frac{f_i}{t_i+f_i}[B_{i,f} - B_{i-1}] \\ &= \frac{t_i^2 + f_i^2}{t_i+f_i}, \end{aligned}$$

while by Lemma 3.4

$$\mathbb{E}[w(\mathrm{OPT}_{i-1}) - w(\mathrm{OPT}_i)] \le \frac{2t_i f_i}{t_i + f_i}.$$

Therefore in order to verify the inequality, we need to show that when $t_i, f_i > 0$,

$$\frac{2t_i f_i}{t_i + f_i} \le \frac{t_i^2 + f_i^2}{t_i + f_i},$$

which follows since $t_i^2 + f_i^2 - 2t_i f_i = (t_i - f_i)^2 \ge 0$. $\square$

In Appendix A we describe a data structure to implement the algorithm in linear time.

4. A tight bound for Online MAX SAT.

In the previous section we have presented a randomized greedy algorithm that achieves a $\frac{3}{4}$-approximation. Interestingly, this algorithm considers the variables in a fixed order and draws its strength only from carefully chosen assignment probabilities. We wonder if these probabilities can be fine-tuned further to obtain an even better guarantee?

Therefore, we investigate the limitations of priority algorithms for Online MAX SAT, where the variables are given one after another together with the clauses they appear in. Recall that the data item for variable x gives the name of x and the list of undecided clauses x appears in. Further recall that a clause of x is undecided if it has not been satisfied by previous assignments to other variables. For each such clause c the data item reveals the sign of x in c, the weight of c, and the unfixed variables in c, but not their signs.

Upon receiving the data item, the priority algorithm has to make an irrevocable decision for the respective variable. Here it is important that priority algorithms have limited information regarding the input, but no restrictions of time or memory. In this sense, our inapproximability results are "information-theoretic."

We show the following bound.

Theorem 4.1. *For any $\varepsilon > 0$, no randomized priority algorithm achieves a $\frac{3}{4} + \varepsilon$-approximation for Online MAX SAT with high probability (using our data type). Hence, the randomized greedy algorithm is an optimal online algorithm.*

Proof. Let X_b and Y_b for $b \in \{0, 1\}$ denote disjoint sets of n variables each. The clause set is constructed as follows: For each variable pair in $X_b \times Y_b$ there is an equivalence and for each pair in $X_b \times Y_{1-b}$ an inequivalence (see Figure 1). Note that each of these constraints can be represented by two clauses of length two. It is crucial that the algorithm cannot distinguish both constraint types, since our data items do not reveal the signs of neighbors.

The bound follows by an application of Yao's principle [35]. Since we are dealing with an online setting, the adversary decides that the x-variables will precede the y-variables and therefore selects a permutation π on the data items of $X_0 \cup X_1$ uniformly at random. We will show that after all x-variables have been fixed, the algorithm will with high probability be unable to satisfy more than a $\frac{3}{4} + o(1)$ fraction of the clauses, regardless of how the y-variables are set subsequently.

Now assume that the data items of the x-variables arrive according to π: When given the next data item, the (deterministic) algorithm cannot decide whether the current variable belongs to X_0 or X_1, since the clauses given in a data item look identical for all variables in $X_0 \cup X_1$. In particular, no two x-variables share a clause,

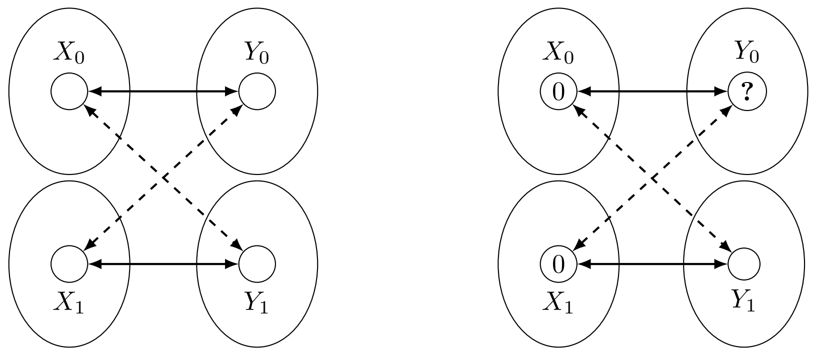

FIG. 1. *Construction for Online MAX SAT (left): For each variable pair in $X_b \times Y_b$ there is an equivalence (solid arrows) and for each pair in $X_b \times Y_{1-b}$ an inequivalence (dashed arrows).*

Example (right) of three assignments to x-variables. Two x-variables were fixed to 0, and no assignment to the variable labeled ? can satisfy both constraints. Observe that every pair of inconsistently fixed x-variables causes a pair of contradicting constraints for each variable in $Y_0 \cup Y_1$.

hence the algorithm cannot observe any effects of its decisions. Therefore, we may assume that first the algorithm fixes all $2n$ x-variables—let Z denote the set of x-variables fixed to zero. Subsequently, the adversary reveals the membership of the x-variables to the sets X_0 and X_1. Note that for any $i \in \{1, \ldots, 2n\}$ the probability that the variable in rank i of π belongs to X_0 (resp., to X_1) is exactly $\frac{1}{2}$. Thus, we have

$$\mathbb{E}\left[|Z \cap X_0|\right] = \mathbb{E}\left[|Z \cap X_1|\right] = \frac{|Z|}{2}$$

by linearity of expectation. In order to prove the stronger claim that the algorithm does not beat the $\frac{3}{4}$-barrier by any constant *with high probability*, we need to show that $|Z \cap X_0|$ and $|Z \cap X_1|$ are concentrated at their mean, i.e., they deviate with high probability by at most $o(n)$ from their expected value. We use a little trick that will allow us to use the nice combinatorial statement we proved in section 2: since the data items are indistinguishable for the algorithm, we may switch things around and assume that the deterministic algorithm sets the variables in the first $|Z|$ positions of the random permutation on $X_0 \cup X_1$ to zero and then the remaining $2n - |Z|$ variables to one. Now we may apply Lemma 2.2 and obtain the number of X_0 (resp., X_1) variables among the first $|Z|$ variables of π deviates from its expectation by more than $\delta \cdot n$ with probability at most $2e^{-\Theta\left(\delta^2 n\right)}$. The claim follows by choosing $\delta = n^{-\varepsilon}$ for some sufficiently small $\varepsilon > 0$.

Now we will count the number of unsatisfied clauses: We call a pair of two x-variables x_i, x_j *contradictory* if the algorithm sets $x_i = x_j$ and they do not belong to the same set X_b (cp. Fig. 1). Therefore, we may form $\min\{|Z \cap X_0|; |Z \cap X_1|\}$ disjoint contradictory pairs among the variables that were fixed to zero by the algorithm. An analogous argument holds for the number of variables fixed to one by the algorithm in X_0 (resp., in X_1): their expected value is $n - \frac{|Z|}{2}$. We also form disjoint contradictory pairs among these variables. Thus, the number of disjoint contradictory pairs is

$$\min\{|Z \cap X_0|; |Z \cap X_1|\} + \min\{|\overline{Z} \cap X_0|; |\overline{Z} \cap X_1|\} \geq n - o(n)$$

with high probability, where $\overline{Z} = X_0 \cup X_1 \setminus Z$.

After fixing all x-variables there will be at least $n - o(n)$ disjoint pairs of contradictory fixed x-variables with high probability, and every such pair is responsible for a single pair of contradicting (in-)equivalences for each variable in $Y_0 \cup Y_1$.

Note that any assignment satisfies at most three out of four clauses for any pair of violated (in-)equivalences. The bound on the approximation ratio follows, since with high probability at least $2n \cdot (n - o(n)) \geq 2n^2 - o(n^2)$ clauses are not satisfied by the algorithm, whereas the optimal assignment obtains all $8n^2$ clauses. ∎

Remark 4.2. We point out that all variables in $X_0 \cup X_1$ must be fixed before the first y-variable is decided. If the algorithm was allowed to fix a single y-variable in advance, it could easily come up with an optimal assignment.

We contrast the performance of deterministic and randomized online algorithms. Assume that the adversary initially presents the clauses

$$(\overline{x} \vee y), (x \vee \overline{y})$$

and variable x is to be decided. If the deterministic algorithm fixes $x = 1$, the formula is completed by adding the unit clause $(\overline{y})$. Otherwise the adversary responds by adding (y). Thus, no deterministic online algorithm is better than $\frac{2}{3}$ approximative—and Johnson's algorithm is optimal in its class.

Recall that our randomized greedy algorithm achieves a $\frac{3}{4}$-approximation for our data type that is more restrictive. Thus, we have obtained a separation between randomized and deterministic online algorithms.

Independently of our work, Azar, Gamzu, and Roth [6] considered a more restrictive data type that in particular does not reveal the length of a clause. For this model they prove that no online algorithm can achieve a $\frac{2}{3} + \varepsilon$-approximation for any $\varepsilon > 0$. Note that their result does not contradict Theorem 4.1, since their data items are too restrictive to run our randomized algorithm.

5. Adaptive priority algorithms for MAX SAT.

In this section we study deterministic priority algorithms that have the ability to choose adaptively the next variable to set. We show that no such algorithm can achieve a $\frac{3}{4}$-approximation ratio given our choice of data items. Since our randomized priority algorithm has approximation ratio $\frac{3}{4}$, our bound gives a strict separation between deterministic and randomized algorithms in this model. In sections 6 and 7 we will see how we can achieve a $\frac{3}{4}$-approximation deterministically by looking at two variants of the randomized greedy algorithm that do not fall in this model.

An adaptive priority algorithm is a deterministic algorithm that may choose the order in which variables are to be processed, utilizing any information it has obtained about the input: This includes data items it has seen so far and also data items that are known not to be part of the input. Recall that our data item for variable x gives the name of x and the list of not yet satisfied clauses x appears in: For each such clause c the data item reveals the sign of x in c, the weight of c, and the undecided variables in c, but not their signs. In particular, literals that were set to zero by previous assignments are omitted. We merge identical clauses by adding up their nonnegative weights. Note that clauses do not carry identifiers; thus the algorithm may be unable to decide for some clauses in the current data item if they were contained in a previously shown data item (but if so, they had not been satisfied by the algorithm's assignment; see also below).

For the inapproximability bound we study the "adaptive priority game" which was introduced by Borodin, Nielsen, and Rackoff [9] and extended to related data items by Davis and Impagliazzo [15] and by Borodin, Boyar, Larsen, and Mirmohammadi [8]. It offers a convenient way to prove a bound ρ on the approximation guarantee achievable by any algorithm in a class $\mathcal{C}$ of deterministic algorithms, by showing that an appropriately restricted adversary can create an instance $I_{\mathcal{A}}$ for every algorithm $\mathcal{A} \in \mathcal{C}$ such that the approximation achieved by $\mathcal{A}$ on $I_{\mathcal{A}}$ is at most ρ. For a formal proof of this intuitive connection we refer to [8]. The adaptive priority game is played as follows:

(i) The game starts with the adversary announcing the number n of variables as well as their names. The actual input is kept private.

(ii) In round i the deterministic algorithm submits an ordering π_i on the data items. The adversary picks the smallest data item d according to π_i in its input and hands d to the algorithm. Note that the algorithm has also learned that none of the data items that precede d in π_i are contained in the input.

Then the algorithm makes an instant and irrevocable decision for the variable given in the data item d. This ends the round.

(iii) When the algorithm has fixed all variables, the adversary presents a clause set that is consistent with the data items presented during the game *and* with the data items that were revealed not to be part of the input.

Using our data type, we show that the adversary is able to force any adaptive priority algorithm into an approximation ratio of at most $\frac{\sqrt{33}+3}{12} + o(1) \approx 0.729 < \frac{3}{4}$.

THEOREM 5.1. *For any $\varepsilon > 0$, no* adaptive *priority algorithm can approximate MAX SAT within $\frac{\sqrt{33}+3}{12} + \varepsilon$ (using our data type).*

Before we give the proof, we compare our work to two related results: Utilizing a stronger data type that reveals all signs of neighboring variables, Yung [36] considers the version of MAX SAT where each clause contains *exactly* two literals and shows that no adaptive priority algorithm can obtain an approximation ratio better than $\frac{5}{6}$. Note that Johnson's algorithm guarantees a $\frac{3}{4}$-approximation in this case.

The result of Alekhnovich et al. [1] (for priority branching trees, using the same stronger data type) implies that no *fixed* priority algorithm can approximate MAX SAT within $\frac{21}{22} + \varepsilon$. Note that fixed priority algorithms must stick with an a priori ordering of potential data items.

Outline of the proof of Theorem 5.1. The adversary will construct a 2CNF formula ϕ with n variables. The overall structure of the formula will have either an equivalence or an inequivalence between each pair of variables; recall that each of these constraints is represented by two clauses of length two. Also, each variable x will have a positive unit clause (x) and a negative unit clause $(\overline{x})$; these clauses will have an integral weight between 0 and G (inclusive), where the integer G will be chosen later. As the game progresses, the adversary will determine whether there is an equivalence or an inequivalence between pairs of variables x and y, as well as the weights of the unit clauses (x) and $(\overline{x})$. The adversary utilizes the fact that our data items do not allow the algorithm to distinguish equivalences from inequivalences. In particular, when the algorithm considers the data item corresponding to a variable x, it is unable to tell whether it has an equivalence or an inequivalence between x and all not yet set variables.

Furthermore, the data item corresponding to variable x only tells the algorithm the total weight of the *current* unit clauses (x) and $(\overline{x})$, and the algorithm is unable to distinguish whether these unit clauses were unit clauses in the original input, or whether they were 2-clauses and hence are the result of having previously set one variable so as to falsify a literal of the clause. We give an example: assume that the algorithm previously fixed a variable y to zero and recall that by construction y shared two 2-clauses with an unfixed variable x. If the algorithm subsequently sees a unit clause (x) of weight one in the data item of variable x, it is unable to tell whether the unit clause was part of the original input, or whether it stems from the clause $(x \vee y)$.

Thus, observe that data items differ only in the name of the respective variable x and in the weights of the two unit clauses (x) and $(\overline{x})$. As a shorthand, we will use the notation (ℓ, ℓ') to refer to the weights of the two unit clauses of some variable x (also called the "unit weights" of x), where ℓ is a natural number that gives the weight of the positive unit clause (x) and ℓ' is the weight of the negative unit clause $(\overline{x})$ (both after merging identical clauses). We will call (ℓ, ℓ') the weights of the *resulting* unit clauses for x, to distinguish these weights from the weights of the *original* unit clauses of the input.

The class of adaptive priority algorithms admits a large variety of algorithmic strategies, hence the description of the adversary also requires some flexibility. We call data items "feasible" or "allowed" by the adversary if they may occur in the input; whether a feasible data item actually is part of the instance depends on the algorithm: given the ordering π on the data items that were submitted by the algorithm at the beginning of the round, the adversary picks the first feasible data item according to π and hands it to the priority algorithm.

The game is divided into two phases. Phase (I) lasts for $G < n$ rounds. We set a trap for the algorithm by introducing contradictions for the variables of phase (II) in each round.

In round $k \in \{1, 2, \ldots, G\}$ (of phase (I)) the adversary allows data items for the remaining variables only if the weights of both unit clauses are at least $k - 1$ and at most G. No other data items are feasible. In particular, the combinations $(G, 0)$, $(0, G)$ of unit weights are feasible at the beginning of round 1, a fact that will be used for the variables of phase (II).

Phase (II) consists of $(n - G)$ rounds. Here we make sure that the algorithm does not recover. At the beginning of phase (II), in round $G + 1$, only data items of variables with weight combination (G, G) will be feasible. In particular, we will show that the adversary can enforce the resulting unit clauses of these variables to have weights (G, G), starting from original unit clauses in the input that have weight combinations $(G, 0)$ or $(0, G)$.

From now on, the adversary observes two consecutive rounds, i.e., two decisions of the algorithm on variables x and y, and then decides whether x and y had equivalences or inequivalences with previously set variables, as well as the weights of the original unit clauses on x and y. W.l.o.g. assume that $(n-G)$ is even. In round $G+2m+1$ the resulting weights of the unit clauses will be $(G+m, G+m)$, and in round $G+2m+2$ the resulting weights of the unit clauses will be $(G+m+1, G+m)$ or $(G+m, G+m+1)$, where $m \in \{0, 1, \ldots, \frac{n-G}{2} - 1\}$ holds.

Without knowing ϕ exactly, we can give an upper bound on the total weight that the algorithm can satisfy. In order to do this, we take its perspective and look at the clauses it has satisfied, i.e., as they appear in the data items given to the algorithm. Regardless of the value it assigns to the variable, the algorithm will satisfy exactly one clause of each equivalence, resp., of each inequivalence. The other clause of each (in-)equivalence is not satisfied in this round and hence appears again as a unit clause. It will be counted only if the algorithm satisfies the unit clause at some later time.

Thus, the total weight of satisfied clauses of length two equals

$$\sum_{k=1}^{n} [n-k] = \frac{n^2-n}{2}. \tag{5.1}$$

Moreover, the algorithm may always satisfy the unit clause of larger weight: The total weight of unit clauses satisfied in phase (I) is at most G^2, since G is an upper bound on the weight of any resulting unit clause during phase (I). In phase (II) unit clauses of total resulting weight at most

$$\begin{aligned}&\sum_{m=0}^{\frac{n-G}{2}-1} [(G+m)+(G+m+1)]\\ &= (2G+1)\cdot\frac{n-G}{2} + \frac{2}{2}\cdot\frac{n-G}{2}\cdot\left(\frac{n-G}{2}-1\right)\\ &= \frac{n^2}{4} + \frac{1}{2}nG - \frac{3}{4}G^2\end{aligned}$$

are satisfied. Thus, the total weight of clauses satisfied by the algorithm is at most

$$\begin{aligned}\mathrm{Sat} &= \frac{n^2-n}{2} + G^2 + \frac{n^2}{4} + \frac{1}{2}nG - \frac{3}{4}G^2\\ &= \frac{3}{4}n^2 + \frac{1}{2}nG + \frac{1}{4}G^2 - \frac{n}{2}.\end{aligned} \tag{5.2}$$

We will show that the adversary is able to construct an almost satisfiable formula.

Proof of Theorem 5.1. We begin by describing the adversary in phase (I).

The adversary's strategy in phase (I). Assume that the algorithm processes variable x in round k. Let y be some variable set by the algorithm in a previous round. The key insight of the construction is that the adversary may determine whether there is an equivalence or an inequivalence between x and y *after* the algorithm sets the variable x (that is, after the algorithm has set both x and y!). In order to see this, we consider the algorithm's perspective on the equivalence between y and x, i.e., $(y \vee \overline{x}), (\overline{y} \vee x)$, where y was fixed before round k. As pointed out above, the information given in the data items does not allow the algorithm to distinguish between equivalences and inequivalences; hence after fixing y the algorithm knows only that a unit clause for x was created, but it does not know the sign of the new unit clause, nor does it know the weight of the unit clauses (x) and $(\overline{x})$ that were in the original input. Thus, if the feasible data items of x each have unit clauses of sufficiently large weight *and* this was also true in the previous rounds (accounting for the fact that in each round a new unit clause was created for x), then the adversary can decide *after the algorithm set* x whether the unit clause (x) and $(\overline{x})$ resulted from an (in-)equivalence with a previously set variable y or from the original input. The reason is that the data item presented to the algorithm is identical in either case.

In round k of phase (I), the adversary will assert to the algorithm that all data items associated with variables not yet set have resulting unit weights of at least $k-1$ and at most G; that is, no other data items are feasible.

Now let us describe the concrete strategy of the adversary during phase (I). Let Z_0 (Z_1) denote the variables that the algorithm has fixed to zero (to one, resp.) at the beginning of round k. If the algorithm sets variable x to zero, the adversary decides that there were equivalences between x and all variables in Z_0 and inequivalences between x and all variables in Z_1. Note that an equivalence between a variable y set to zero and x results in a unit clause $(\overline{x})$, while an inequivalence between a variable y set to one and x also results in a unit clause $(\overline{x})$. Thus, in each of the $k-1$ previous rounds a new unit clause $(\overline{x})$ (of weight one) was created. What is the weight of the original unit clause $(\overline{x})$ of the input? If $g \in \{k-1, k, \ldots, G\}$ is the weight of the resulting clause $(\overline{x})$ in round k, then the weight of the original clause is set to $g-k+1$. In particular, this weight is nonnegative as we required.

If x is set to one, the argument is analogous.

The assignment of the adversary for the phase (I) variables is obtained by flipping the assignment of the algorithm. As a consequence, the adversary also satisfies all (in-)equivalences between variables fixed in phase (I).

The feasibility of the data item of x in previous rounds. Now we argue that the data item of x was feasible in all previous rounds. Therefore, the adversary plays "fair," and then in particular the reason why the data item of x was not given to the algorithm in an earlier round is that another data item had higher priority according to the respective ordering submitted by the algorithm. Recall that in round j a data item was feasible only if the weights of the resulting unit clauses were at least $j-1$ (and at most G).

Again assume that x is set to zero (the other case follows analogously) and note that the resulting unit clause $(\overline{x})$ met this requirement in each round for the adversary's choice of the original weight. Furthermore, we have seen that the 2-clauses that x occurred in together with variables of Z_0 and Z_1 do not result in new unit clauses (x). Thus, the weight of the resulting unit clause (x) in round k equals the weight of the original clause (x) in the input; by construction, this weight is at least $k-1$ and at most G for the data item of round k; therefore the weight also met the requirement in all previous rounds. Summing up, the data item of x that the algorithm received in round k was feasible in all previous rounds.

The adversary's strategy for phase (II). Now we describe the adversary's strategy for the variables that the algorithm fixes during phase (II). The adversary asserts to the algorithm that all remaining (i.e., still unset) variables have resulting unit clause weights (G, G) at the beginning of phase (II), that is in round $G+1$. The adversary can now do either of the following for each remaining variable x: (1) It can decide that the original unit clause (x) in the input had weight G, while all the weight G of unit clause $(\overline{x})$ resulted from equivalences with all variables in Z_0 and inequivalences with all variables in Z_1; we will say that the adversary declares these variables to have *original* unit clause weights $(G, 0)$. Or (2) it can decide the reverse, namely that G of the weight of the resulting unit clause $(\overline{x})$ was from the original input, while all the weight G of the unit clause (x) resulted from inequivalences with all variables in Z_0 and equivalences with all variables in Z_1. In this case, we will say that the adversary declares these variables to have original unit clause weights $(0, G)$. Thus, the adversary can decide for each phase (II) variable whether it had original unit weights $(G, 0)$ or $(0, G)$. The adversary's assignment will always satisfy the original unit clause of weight G.

Moreover, we observe that every phase (II) variable had either resulting unit weights $(G, k-1)$ or $(k-1, G)$ at the beginning of round k in phase (I). In either case the data item of the phase (II) variable was feasible in that round.

Deciding the constraints between phase (II) variables. During phase (II) the adversary observes two consecutive assignments x, y made by the algorithm. Recall that $n - G$ was assumed to be even and that $m \in \{0, 1, \dots, \frac{n-G}{2} - 1\}$ holds. Let us assume inductively that the adversary only allows variables with resulting unit weights $(G + m, G + m)$ at the beginning of round $G + 2m + 1$. We have already verified that this can be achieved for the base case $m = 0$. For the inductive step assume w.l.o.g. that the algorithm fixes variable x to one in round $G + 2m + 1$. Then the algorithm considers variable y in round $G + 2m + 2$; we consider cases below.

Case 1: Variable y has resulting weight combination $(G+m+1, G+m)$ and the algorithm fixes variable y to zero.

Observe that y had resulting unit weights $(G + m, G + m)$ at the beginning of round $G+2m+1$ and the resulting weight of unit clause (y) increased by one according to the case assumption. To achieve this increase, the adversary decides that there is an equivalence between x and y. Hence, y indeed has weight combination $(G + m + 1, G + m)$ at the beginning of round $G + 2m + 2$, since the algorithm fixed x to one. Moreover, for the adversary's assignment, it sets both variables to one; then it decides there are equivalences between x, y and all variables of phase (II) that it has set to one and inequivalences between x, y and all variables of phase (II) that it has set to zero. Finally, the adversary decides that x and y had original unit weights $(G, 0)$, which determines the constraints with phase (I) variables. Note that the unit clause of weight G is satisfied by the adversary for both variables.

Unlike the adversary, the algorithm sets variables x and y differently. Any currently unset variable z will be connected with x and y via equivalences only or inequivalences only. Hence both unit weights of z increase by one and we can conclude the inductive argument.

Case 2: Variable y has resulting weight combination $(G+m+1, G+m)$ and the algorithm fixes variable y to one. In the adversary's assignment x is set to one and y to zero.

Observe that y has resulting unit weights $(G + m, G + m)$ at the beginning of round $G + 2m + 1$ and the weight of unit clause (y) increases by one according to the case assumption. Again the adversary decides that there is an equivalence between x and y but in this case satisfies only one out of the two clauses of that constraint. Note that y indeed has weight combination $(G + m + 1, G + m)$ at the beginning of round $G + 2m + 2$. This time, the adversary decides that there are equivalences between $x, \overline{y}$ and all variables of phase (II) that it has set to one and inequivalences between $x, \overline{y}$ and all variables of phase (II) that it has set to zero.

For variable x the adversary decides that its original unit weights are $(G, 0)$, whereas y receives the original unit weights $(0, G)$. Again the adversary's assignment satisfies the unit clauses of original weight G.

Unlike the adversary, the algorithm set x and y identically. Any currently unset variable z is later connected with x and $\overline{y}$ via equivalences only or inequivalences only. Hence both unit weights of z increase by one. Thus we can conclude the inductive argument in this case as well.

The cases that variable y has weight combination $(G+m, G+m+1)$ in round $G+2m+2$ are treated analogously.

We now calculate the total weight of clauses satisfied by the adversary's assignment. Recall that the adversary constructed the (in-)equivalences in a way such that for each pair of phase (I) variables the respective constraint is satisfied by the adversary. The same holds for each pair consisting of a phase (I) variable and a phase (II) variable. Moreover, we argued that only at most $\frac{n-G}{2}$ (in-)equivalences among pairs of two phase (II) variables are falsified, i.e., at most one for every two consecutive rounds in phase (II). Hence the adversary satisfies at least $\binom{n}{2} - \frac{n-G}{2}$ (in-)equivalences, i.e., at least $n\cdot(n-1) - \frac{n-G}{2}$ 2-clauses of weight one. To see this, recall that each (in-)equivalence is represented as two 2-clauses of weight one each, and for a unsatisfied (in-)equivalence constraint only one out of two 2-clauses is satisfied.

Recall that for every phase (II) variable the adversary satisfies the unit clause of original weight G, and thus the adversary obtains unit clauses of phase (II) variables with a combined weight of $(n - G) \cdot G$.

Now consider the variable x fixed in round k of phase (I). Remember that if x was to zero by the algorithm, the adversary decided that there were equivalences between x and all variables in Z_0 as well as inequivalences between x and all variables in Z_1. Thus, only new unit clauses $(\overline{x})$ were created up to that point, and in particular the resulting weight of (x) equals its original weight in the input. Recall the data item given in round k had resulting unit clauses of weight of at least $k - 1$. Because the adversary's assignment was the inverse of the algorithm's assignment, the adversary satisfies this unit clause (x) with original weight at least $k - 1$.

The case that variable x was set to one by the algorithm is analogous. Thus, the weight of satisfied unit clauses for phase (I) variables is at least $\sum_{k=1}^{G} [k-1] = \sum_{k=1}^{G-1} k = \frac{G\cdot(G-1)}{2}$. Summing up, the assignment of the adversary satisfies clauses of total weight at least

$$\begin{aligned} & n\cdot(n-1) - \frac{(n-G)}{2} + (n-G)\cdot G + \frac{G\cdot(G-1)}{2} \\ (5.3)\qquad & = n^2 + nG - \frac{1}{2}G^2 - \frac{3}{2}n. \end{aligned}$$

Combining (5.2) and (5.3) and then choosing $G := \alpha \cdot n$ yields

$$\frac{\text{Sat}}{\text{Opt}} \leq \frac{\frac{3}{4}n^2 + \frac{1}{2}nG + \frac{1}{4}G^2 - \frac{n}{2}}{n^2 + nG - \frac{1}{2}G^2 - \frac{3}{2}n} = \frac{\frac{3}{4} + \frac{1}{2}\alpha + \frac{1}{4}\alpha^2 - o(1)}{1 + \alpha - \frac{1}{2}\alpha^2 - o(1)}$$

as an upper bound on the approximation ratio. A simple calculation shows that the ratio is minimized for $\alpha = \frac{\sqrt{33}-5}{4}$ and that the approximation ratio converges to at most $\frac{\sqrt{33}+3}{12}$. ∎

6. A deterministic LP rounding scheme. One way to overcome our inapproximability result for adaptive priority algorithms is to reveal more information about the variable the algorithm has to decide. In this section we consider the scenario that the algorithm is additionally given the optimal value according to the canonical LP (see section 3). Then we can take essentially the algorithm and analysis given in section 3 and use it to obtain a deterministic LP rounding algorithm.

Note that this gives an LP rounding scheme that in order to decide a variable x_i considers only the clauses that x_i appears in and the optimal LP-value for x_i. Derandomization of the rounding schemes of Goemans and Williamson [18] via the method of conditional expectations would also require the LP-values for unfixed variables that appear in the clauses of x_i.

Let y^* be an optimal solution to the standard LP relaxation. As before, our algorithm will sequence through the variables x_i, deciding at each step whether to set x_i to true or false; now the decision will be made deterministically. Let B_i be the same bound as before, and as before let t_i be the increase in the bound if x_i is set true, and f_i the increase if x_i is set false. Let LP_i be the value of the LP for a vector

$\hat{y}$ in which the first i elements are zeros and ones corresponding to our assignment S_i, while the remaining entries are the values of the optimal LP solution $y^*_{i+1}, \ldots, y^*_n$. Thus $\mathrm{LP}_0 = \mathrm{OPT}_{\mathrm{LP}}$ and $\mathrm{LP}_n = w(S_n)$, the weight of our assignment. We further introduce the notation $\mathrm{LP}_{i,t}$ ($\mathrm{LP}_{i,f}$), which correspond to the value of the LP for the vector $\hat{y}$ in which the first $i-1$ elements are zeros and ones corresponding to our assignment S_{i-1}, the entries for $i+1$ to n are the values of the optimal LP solution $y^*_{i+1}, \ldots, y^*_n$, and the ith entry is one (zero, resp.). Note that after we decide whether to set x_i true or false, either $\mathrm{LP}_i = \mathrm{LP}_{i,t}$ (if we set x_i true) or $\mathrm{LP}_i = \mathrm{LP}_{i,f}$ (if we set x_i false).

The algorithm is then as follows. When we consider variable x_i, we check whether $2y^*_i t_i \le f_i$: if the inequality holds, we set x_i false (and thus $\mathrm{LP}_i = \mathrm{LP}_{i,f}$); otherwise we set x_i true (and thus $\mathrm{LP}_i = \mathrm{LP}_{i,t}$).

The following lemma is analogous to Lemma 3.2.

Lemma 6.1. *For $i = 1, \ldots, n$,*

$$\mathrm{LP}_{i-1} - \mathrm{LP}_i \le B_i - B_{i-1}.$$

Proof. If $2y^*_i t_i \le f_i$, we set x_i to false, so that $\mathrm{LP}_{i-1} - \mathrm{LP}_i = \mathrm{LP}_{i-1} - \mathrm{LP}_{i,f}$ and $B_i - B_{i-1} = f_i$. Equation (3.2) from the proof of Lemma 3.4 gives an upper bound on the change in the objective value: we obtain $\mathrm{LP}_{i-1} - \mathrm{LP}_{i,f} \le 2y^*_i t_i$, and by the condition for setting x_i to false, this is at most f_i.

If $2y^*_i t_i > f_i$, we set x_i to true, so that $\mathrm{LP}_{i-1} - \mathrm{LP}_i = \mathrm{LP}_{i-1} - \mathrm{LP}_{i,t}$ and $B_i - B_{i-1} = t_i$. Now we utilize (3.1) from the proof of Lemma 3.4 to obtain $\mathrm{LP}_{i-1} - \mathrm{LP}_{i,t} \le 2(1 - y^*_i) f_i$. Since $f_i < 2y^*_i t_i$, this is less than $4 \cdot (1 - y^*_i) \cdot y^*_i \cdot t_i \le t_i$, where the final inequality uses the fact that $0 \le y^*_i \le 1$. ∎

Given the lemma, we can prove the following.

Theorem 6.2. *Let* OPT *be the total weight satisfied by an optimal Boolean assignment and denote by* OPT_{LP} *the objective value of the LP relaxation.*

For the assignment S_n computed by the algorithm

$$w(S_n) \ge \frac{1}{2}\mathrm{OPT}_{\mathrm{LP}} + \frac{W}{4} \ge \frac{3}{4}\mathrm{OPT}$$

holds.

Proof. As in the proof of Theorem 3.3, we sum together the inequalities given by Lemma 6.1, so that

$$\sum_{i=1}^{n} (\mathrm{LP}_{i-1} - \mathrm{LP}_i) \le \sum_{i=1}^{n} (B_i - B_{i-1}).$$

Telescoping the sums, we get

$$\mathrm{LP}_0 - \mathrm{LP}_n \le B_n - B_0, \quad \text{or} \quad \mathrm{OPT}_{\mathrm{LP}} - w(S_n) \le w(S_n) - \frac{W}{2}.$$

Rearranging terms, we have

$$w(S_n) \ge \frac{1}{2}\mathrm{OPT}_{\mathrm{LP}} + \frac{W}{4} \ge \frac{3}{4}\mathrm{OPT},$$

since both $\mathrm{OPT}_{\mathrm{LP}} \ge \mathrm{OPT}$ (note that the LP is a relaxation) and $W \ge \mathrm{OPT}$ hold. ∎

7. A deterministic 2-pass algorithm.

Why is it not possible to derandomize the randomized greedy algorithm (resp., any of its variants) via the method of conditional expectations? The problem is that the assignment probabilities used in each step depend on the outcome of previous decisions, since these determine which clauses are undecided, and only undecided clauses are taken into consideration in later rounds.

In this section we present a simple, deterministic $\frac{3}{4}$-approximation algorithm that performs two passes over the variables in a fixed order. In the first pass the algorithm behaves essentially as our randomized greedy algorithm: it computes an "assignment probability" for each variable but does not fix the variables yet. Then in a second pass the algorithm obtains a Boolean assignment deterministically.

The gist is that in order to compute the probability for variable x_i in the first pass, we pretend that the variables $x_1, \dots, x_{i-1}$ were fixed independently according to their respective assignment probabilities. Therefore, in the second pass we may invoke the method of conditional expectations to derandomize the random assignment that was computed in the first pass.

The idea of deferring the decisions about the variables was used by Buchbinder et al. [10]: they propose the *fractional greedy algorithm* for the unconstrained submodular maximization problem (USM). In this problem one is given a normalized, nonnegative, submodular function f over a ground set X and wants to find a subset $S \subseteq X$ that maximizes $f(S)$. The fractional greedy algorithm makes a single pass over the ground set and computes for each element i a probability p_i, but defers the decision whether to include the element in S. After computing all probabilities, each element is picked independently with probability p_i. Buchbinder et al. [10] show that this gives an expected $\frac{1}{2}$-approximation for USM.

7.1. The analysis of the 2-pass algorithm.

First we show that the deterministic 2-pass algorithm achieves the same worst-case guarantee as the randomized greedy algorithm given in section 3. Then we will detail in section 7.2 how to implement the algorithm efficiently. In sections 7.3 and 7.4 we study the question whether the additional pass over the input can give a better overall performance than the randomized greedy algorithm that utilizes a single pass only.

7.1.1. The computation tree perspective.

We begin with visualizing the randomized greedy algorithm by the following "computation tree": for a clause set on the variables $x_1, \dots, x_n$ this is the complete binary tree of depth n. Every node in depth i is labeled with a partial assignment $\vec{b}_i \in \{0,1\}^i$ to $x_1, \dots, x_i$ and connected to the two unique nodes that extend the partial assignment $\vec{b}_i$ by fixing x_{i+1} to zero or to one, respectively. In particular, the root corresponds to the empty assignment and each leaf gives a complete assignment to all n variables. The edges of the computation tree are labeled by the respective assignment probabilities used by the randomized greedy algorithm. Recall that the assignment probabilities for x_{i+1} depend only on the clauses that x_{i+1} appears in and the respective partial assignment to $x_1, \dots, x_i$.

Now we imagine the randomized greedy algorithm as a player that starts in the root and then descends by choosing iteratively a child at random, using the assignment probabilities as "transition probabilities." Therefore, we obtain a distribution over the nodes of the tree, where the probability that the player visits a node v equals the product of the transition probabilities on the path from v to the root. In particular, Theorem 3.3 implies that the player ends in a leaf whose corresponding assignment satisfies clauses with a total weight of at least $\frac{1}{2} \cdot \mathrm{OPT}_{LP} + \frac{1}{4} \cdot W$ *in expectation.*

For the 2-pass algorithm we give a computation tree with the same structure, but the transition probabilities are set differently: for the nodes of depth $i-1$ all transition probabilities that set $x_i = 1$ equal the same $\sigma_i \in [0,1]$, and thus all the transitions $x_i = 0$ are taken with probability $1-\sigma_i$. Then the first pass of the 2-pass algorithm can be modeled by a player that behaves exactly as the player corresponding to the randomized greedy algorithm, i.e., starting with the root it iteratively picks a child at random according to the decision probabilities and descends. Note the fact that the 2-pass algorithm defers fixing variables can be illustrated naturally by allowing its player to be simultaneously in several (or all) nodes of the current depth during the first pass: the "extent" of the player being in some node v equals its probability of visiting v.

7.1.2. The randomized greedy algorithm revisited. The analysis of the randomized greedy algorithm presented in section 3 contrasts the expected increase in the total weight of satisfied and unsatisfied clauses by the change in the score of the *reference assignment*. That is, we pick an optimal solution $y^* \in [0,1]^n$ to the canonical LP relaxation of MAX SAT and study how the objective value of the LP changes when the variables are set one-by-one as by the randomized greedy algorithm instead of as in y^*. For each partial assignment $\vec{b}_i$, let $\mathrm{OPT}(\vec{b}_i)$ denote this objective value for the assignment that sets the first i variables as in $\vec{b}_i$ and the others as in y^*; we attach with every node its respective $\mathrm{OPT}(\cdot)$-value. Note that the values are the same for both algorithms.

Moreover, for any node $\vec{b}_{i-1}$ let $\mathrm{SAT}(\vec{b}_{i-1})$ (resp., $\mathrm{UNSAT}(\vec{b}_{i-1})$) denote the total weight of clauses satisfied (resp., unsatisfied) by the partial assignment $\vec{b}_{i-1}$. Recall that Lemma 3.2 states for *every* node $\vec{b}_{i-1}$ with $1 \le i \le n$ (i.e., all nodes except the leaves) that

$$\mathbb{E}\left[B_i - B_{i-1}\right] \ge \mathbb{E}\left[\mathrm{OPT}_{i-1} - \mathrm{OPT}_i\right], \tag{7.1}$$

where the expectation is taken over the random assignment of the randomized greedy algorithm. Let $p(\vec{b}_{i-1})$ denote the probability that the randomized greedy algorithm assigns $x_i = 1$ if it has obtained the partial assignment $\vec{b}_{i-1}$: then we can rewrite (7.1) as

$$\begin{aligned}
&\frac{1}{2}\cdot\Big[p(\vec{b}_{i-1})\cdot\Big(\mathrm{SAT}(\vec{b}_{i-1}\wedge(x_i=1)) + W - \mathrm{UNSAT}(\vec{b}_{i-1}\wedge(x_i=1))\Big)\\
&\quad + \Big(1-p(\vec{b}_{i-1})\Big)\cdot\Big(\mathrm{SAT}(\vec{b}_{i-1}\wedge(x_i=0)) + W - \mathrm{UNSAT}(\vec{b}_{i-1}\wedge(x_i=0))\Big)\\
&\quad - \Big(\mathrm{SAT}(\vec{b}_{i-1}) + W - \mathrm{UNSAT}(\vec{b}_{i-1})\Big)\Big]\\
&\ge \mathrm{OPT}(\vec{b}_{i-1}) - p(\vec{b}_{i-1})\cdot\mathrm{OPT}(\vec{b}_{i-1}\wedge(x_i=1))\\
&\quad - \Big(1-p(\vec{b}_{i-1})\Big)\cdot\mathrm{OPT}(\vec{b}_{i-1}\wedge(x_i=0)),
\end{aligned} \tag{7.2}$$

where $\vec{b}_{i-1}\wedge(x_i=1)$ is the assignment where the nodes in $x_1,\dots,x_{i-1}$ are set as in $\vec{b}_{i-1}$ and x_i is set to one, and $\vec{b}_{i-1}\wedge(x_i=0)$ is defined analogously.

We stress that inequality (7.2) is only valid for an appropriately chosen $p(\vec{b}_{i-1})$ that depends on the respective $\vec{b}_{i-1}$ and may differ across the nodes of the same depth.

In particular, recall that in order to compute $p(\vec{b}_{i-1})$ for a fixed $\vec{b}_{i-1}$, we first calculate

$$\begin{aligned} t_i &= \frac{1}{2} \cdot (\mathrm{SAT}_{i,t} - \mathrm{SAT}_{i-1} + \mathrm{UNSAT}_{i-1} - \mathrm{UNSAT}_{i,t}) \\ &= \frac{1}{2} \cdot \Big(\mathrm{SAT}(\vec{b}_{i-1} \wedge (x_i = 1)) - \mathrm{SAT}(\vec{b}_{i-1}) + \mathrm{UNSAT}(\vec{b}_{i-1}) \\ &\qquad - \mathrm{UNSAT}(\vec{b}_{i-1} \wedge (x_i = 1))\Big) \end{aligned}$$

and

$$\begin{aligned} f_i = \frac{1}{2} \cdot \Big(&\mathrm{SAT}(\vec{b}_{i-1} \wedge (x_i = 0)) - \mathrm{SAT}(\vec{b}_{i-1}) + \mathrm{UNSAT}(\vec{b}_{i-1}) \\ &- \mathrm{UNSAT}(\vec{b}_{i-1} \wedge (x_i = 0))\Big), \end{aligned}$$

and then set $p(\vec{b}_{i-1}) = 1$ if $f_i \leq 0$ holds, and else if $t_i \leq 0$ we set $p(\vec{b}_{i-1}) = 0$. Otherwise we let $p(\vec{b}_{i-1}) = \frac{t_i}{t_i+f_i}$.

7.1.3. The main invariant for the 2-pass algorithm. For the 2-pass algorithm, however, we have to determine a single assignment probability σ_i, since all variables are to be set independently and hence σ_i must not depend on the actual partial assignment $\vec{b}_{i-1}$. Therefore, there may not be a single σ_i such that the invariant (7.2) holds for all $\vec{b}_{i-1}$ when $p(\vec{b}_{i-1})$ is replaced by σ_i: for some of the possible random assignments to $x_1, \ldots, x_{i-1}$ the required invariant will be violated.

Thus, we will relax the requirement and show only that the invariant holds "in expectation." Let "$\vec{x}_{i-1}{=}\vec{b}_{i-1}$" be the event that the partial assignment to $x_1, \ldots, x_{i-1}$ obtained by the 2-pass algorithm is consistent with $\vec{b}_{i-1} \in \{0,1\}^{i-1}$. Then we may define

$$\mathrm{ESAT}_{i-1} = \sum_{\vec{b}_{i-1} \in \{0,1\}^{i-1}} \Big[\mathrm{SAT}(\vec{b}_{i-1}) \cdot \mathrm{prob}\Big[\vec{x}_{i-1}{=}\vec{b}_{i-1}\Big]\Big].$$

The intuition behind this definition of ESAT_{i-1} is that we contract all nodes of depth $i-1$ into a single "super-node," where each of these nodes is represented to the extent that equals its probability of being visited by the 2-pass algorithm player. Thus, ESAT_{i-1} can be seen as the "SAT" value of the super-node. EUNSAT_{i-1} and EOPT_{i-1} are defined analogously; note that EOPT_{i-1} equals the expected value of the reference assignment after fixing the first $i-1$ variables. In particular, we observe that $\mathrm{EOPT}_n = \mathrm{ESAT}_n$ and $\mathrm{EOPT}_0 = \mathrm{OPT}_{LP}$, i.e., it equals the objective value of y^* for the canonical LP.

Next we merge all nodes of depth i corresponding to $x_i{=}1$ and let

$$\mathrm{ESAT}_{i,t} = \sum_{\vec{b}_{i-1} \in \{0,1\}^{i-1}} \Big[\mathrm{SAT}(\vec{b}_{i-1} \wedge (x_i = 1)) \cdot \mathrm{prob}\Big[\vec{x}_{i-1}{=}\vec{b}_{i-1}\Big]\Big].$$

Again, the definitions of $\mathrm{EUNSAT}_{i,t}$ and $\mathrm{EOPT}_{i,t}$ are analogous. Finally, we contract all nodes of depth i where $x_i = 0$ into a single node to obtain $\mathrm{ESAT}_{i,f}$, $\mathrm{EUNSAT}_{i,f}$, and $\mathrm{EOPT}_{i,f}$.

Let us assume for a moment that we would have an assignment probability σ_i such that the following analogue of invariant (7.2) held, where each quantity of the randomized greedy algorithm is replaced by its respective analog for the 2-pass algorithm:

$$\begin{aligned}&\frac{1}{2}\cdot[\sigma_i\cdot(\mathrm{ESAT}_{i,t}-\mathrm{EUNSAT}_{i,t})+(1-\sigma_i)\cdot(\mathrm{ESAT}_{i,f}-\mathrm{EUNSAT}_{i,f})\\ &\qquad-(\mathrm{ESAT}_{i-1}-\mathrm{EUNSAT}_{i-1})]\\ &\geq\mathrm{EOPT}_{i-1}-[\sigma_i\cdot\mathrm{EOPT}_{i,t}+(1-\sigma_i)\cdot\mathrm{EOPT}_{i,f}]\,.\end{aligned}\tag{7.3}$$

We show that this would imply the desired overall guarantee. First observe that $\mathrm{EOPT}_i=\sigma_i\cdot\mathrm{EOPT}_{i,t}+(1-\sigma_i)\cdot\mathrm{EOPT}_{i,f}$ holds by definition; thus the right-hand side of (7.3) equals $\mathrm{EOPT}_{i-1}-\mathrm{EOPT}_i$. Similar to the proof of Theorem 3.3, summing up the invariant (7.3) for all $i\in\{1,\dots,n\}$ gives

$$\begin{aligned}\frac{1}{2}\cdot(\mathrm{ESAT}_n-\mathrm{EUNSAT}_n)&\geq\mathrm{EOPT}_0-\mathrm{EOPT}_n,\\ \frac{1}{2}\cdot(\mathrm{ESAT}_n-\mathrm{EUNSAT}_n+W)&\geq\mathrm{EOPT}_0-\mathrm{EOPT}_n+\frac{W}{2},\\ 2\cdot\mathrm{ESAT}_n&\geq\mathrm{OPT}_{LP}+\frac{W}{2},\end{aligned}$$

where we used $\mathrm{EOPT}_n=\mathrm{ESAT}_n$ and $\mathrm{ESAT}_n+\mathrm{EUNSAT}_n=W$. Thus, fixing each variable x_i independently at random according to its assignment probability σ_i satisfies clauses with an expected total weight of $\mathrm{ESAT}_n\geq\frac{1}{2}\cdot\mathrm{OPT}_{LP}+\frac{1}{4}\cdot W$.

In order to obtain a suitable σ_i, we use that (7.3) is obtained from (7.1) by replacing the quantities of the randomized greedy algorithm by their analogues for the 2-pass algorithm. Thus, we determine the assignment probability σ_i in the same way. First we define

$$t_i=\frac{1}{2}\cdot(\mathrm{ESAT}_{i,t}-\mathrm{ESAT}_{i-1}+\mathrm{EUNSAT}_{i-1}-\mathrm{EUNSAT}_{i,t})$$

and

$$f_i=\frac{1}{2}\cdot(\mathrm{ESAT}_{i,f}-\mathrm{ESAT}_{i-1}+\mathrm{EUNSAT}_{i-1}-\mathrm{EUNSAT}_{i,f})\,,$$

which are the analogues of the t- and f-values used by the randomized greedy algorithm. Then, analogously to how we defined $p(\vec{b}_i)$, we set $\sigma_i=1$ if $f_i\leq 0$ holds; otherwise if $t_i\leq 0$ we set $\sigma_i=0$. In any other case we set $\sigma_i=\frac{t_i}{t_i+f_i}$ as the assignment probability for x_i. Thus, invariant (7.3) holds for this choice of σ_i as shown in Lemma 3.2.

We can derandomize the random assignment, where each variable x_k is set independently according to σ_k, in a second pass via the method of conditional expectations, and hence obtain an assignment whose total clause weight is at least as good. Note that this method has a natural interpretation for the computation tree we defined: in each step, starting from the root, we pick the assignment that corresponds to the subtree of larger expected satisfied clause weight. As a convention, we break ties towards the one-assignment.

The performance guarantee for the deterministic 2-pass algorithm is stated in the following theorem.

THEOREM 7.1. *Let* $\mathrm{OPT}_{\mathrm{LP}}$ *denote the optimal objective value of the standard LP relaxation and let* W *be the total weight of all clauses.*

The first pass of the algorithm computes assignment probabilities σ_i for $1 \le i \le n$, such that fixing each variable x_i randomly and independently according to σ_i satisfies a total expected weight of at least

$$\mathrm{ESAT}_n \ge \frac{2\mathrm{OPT}_{LP} + W}{4} \ge \frac{3}{4}\mathrm{OPT}_{LP}.$$

The second pass of the algorithm obtains deterministically a Boolean assignment that satisfies clauses with a total weight of at least ESAT_n.

Both passes can be implemented in linear time.

Observe that the deterministic 2-pass algorithm is not a priority algorithm according to the definition by Borodin, Nielsen, and Rackoff [9]: since it first determines only an assignment probability for each variable and later, after exploring the whole input, eventually decides the variables, it violates a fundamental requirement of priority algorithms. Therefore, the performance guarantee of Theorem 7.1 does not contradict the inapproximability bound of $0.729 < \frac{3}{4}$ for deterministic priority algorithms shown in section 5.

In the next section we describe how the assignment probabilities can be computed in amortized linear time, before we compare the 2-pass algorithm with the randomized greedy algorithm in section 7.3. Then we demonstrate in section 7.4 that the worst-case guarantee given in Theorem 7.1 is tight.

7.2. An efficient computation of the assignment probabilities. Recall that the assignment probabilities are functions of t_i and f_i. However, t_i and f_i are defined as differences of sums with an exponential number of summands each, hence it remains to show that we can compute them quickly. Indeed, these values can be computed in amortized constant time, using the following equivalent definition of ESAT_{i-1} (and analogously for EUNSAT_{i-1}). For a more concise notation, let R_{i-1} denote a Boolean assignment to $x_1, \ldots, x_{i-1}$, where variable x_k is fixed to one independently with probability σ_k and to zero otherwise for $1 \le k \le i-1$:

$$\mathrm{ESAT}_{i-1} = \sum_{c \in C} w_c \cdot \mathrm{prob}\left[c \text{ is satisfied by } R_{i-1}\right].$$

Then, for example, we can rewrite $\mathrm{ESAT}_{i,t} - \mathrm{ESAT}_{i-1}$ as

$$\sum_{c \in C\colon \text{literal } x_i \in c} w_c \cdot \left(1 - \mathrm{prob}\left[c \text{ is satisfied by } R_{i-1}\right]\right);$$

i.e., we sum only over the clauses that contain the (positive) literal x_i. To compute $\mathrm{ESAT}_{i,f} - \mathrm{ESAT}_{i-1}$, we would sum over the clauses that contain the literal $\overline{x}_i$; the change in the total weight of unsatisfied clauses can be expressed similarly.

The gist is that this probability can be computed in constant time due to the relationship

$$\mathrm{prob}\left[c \text{ is not satisfied by } R_i\right] = (1 - \sigma_i) \cdot \mathrm{prob}\left[c \text{ is not satisfied by } R_{i-1}\right]$$

for a clause c that contains literal x_i; the case of literal $\overline{x}_i$ is analogous. Here we use that the assignment probabilities of the 2-pass algorithm are computed using that each variable is fixed *independently*. Therefore, as we consider one variable after another, it suffices to store for each clause the probability of being not satisfied if all previous variables were set according to their respective assignment probabilities.

The crucial idea for the implementation of the second pass is that we can avoid computing the conditional expectations exactly. Therefore, we achieve an amortized cost that is linear in the number of literals in the input. In Appendix B we describe a suitable data structure that supports the above operations and allows us to compute t_i and f_i in amortized constant time.

7.3. The power of the second pass. To demonstrate the power of a second pass, we consider the family $\mathcal{F}_n$ of 2CNF formulae defined as

$$(x_1 \vee \overline{y}), (\overline{x}_1 \vee y), (x_2 \vee \overline{y}), (\overline{x}_2 \vee y), \ldots, (x_n \vee \overline{y}), (\overline{x}_n \vee y).$$

If the randomized greedy algorithm processes the variables in lexicographical ordering, it will set the x-variables before fixing y.

In what follows we assume that both the randomized greedy algorithm and the deterministic 2-pass algorithm consider the variables in lexicographical ordering. We show the following.

THEOREM 7.2. *On input $\mathcal{F}_n$ the approximation ratio achieved by the randomized greedy algorithm converges to $\frac{3}{4}$ as $n \to \infty$.*

The deterministic 2-pass algorithm satisfies all clauses of $\mathcal{F}_n$ for every n.

Proof. We begin with proving the first claim. Since there are no unit clauses when fixing the x-variables and each such variable occurs once with each sign, each x-variable is fixed to true with probability $\sigma_i = \frac{1}{2}$ independently, and to false otherwise.

Thus, by a standard argument the number of x-variables fixed to one is concentrated at its mean $\frac{n}{2}$, and the approximation ratio achieved by the randomized greedy algorithm is $\frac{3}{4} + o(1)$ with high probability as $n \to \infty$.

To prove the second claim of the theorem, we observe that there is no clause that contains more than one x-literal. Since we assume that the deterministic algorithm processes the variables according to the lexicographical ordering, the value computed in the first pass equals the probability used by the randomized greedy algorithm; hence we have $\sigma_i = \frac{1}{2}$ for x_i.

What value is chosen for y in the first pass? Since all variables it shares clauses with are fixed to $\frac{1}{2}$ and the clause set is symmetric, it is also fixed to $\frac{1}{2}$.

In the second pass the algorithm computes the conditional expectations. By symmetry all are the same, hence the deterministic algorithm fixes all x-variables to the same value. Given the assignment to the x-variables, the y-variable is set to the same value; the deterministic algorithm obtains an optimal assignment. ∎

Note that the random assignment obtained by the 2-pass algorithm at the end of its first pass may be worse (in expectation) than the one given by the randomized greedy algorithm: for example, this phenomenon occurs on the clause set $(x \vee y), (\overline{x} \vee \overline{y})$. Here the randomized greedy algorithm always satisfies both clauses, but the random assignment of the 2-pass algorithm has an expected weight of only $\frac{3}{2}$.

From the computation tree perspective introduced above it is interesting to see how the randomized greedy algorithm prunes certain subtrees that are "locally bad," whereas the 2-pass algorithm has no such flexibility and must keep them alive during the first pass. This is remedied in the second pass, however, and hence the assignment returned at the end of the second pass will satisfy both clauses as well.

7.4. A proof of the tightness of the analysis. Theorem 7.1 gives an approximation guarantee of $\frac{1}{2}\mathrm{OPT}_{LP} + \frac{1}{4}W \geq \frac{3}{4}\mathrm{OPT}_{LP}$ for the deterministic 2-pass algorithm. Is this tight? Or does this variant perform even better than its randomized, single pass counterpart?

This question is also motivated by the following observation: the naive algorithm that uses assignment probability $\frac{1}{2}$ for every variable obtains only a $\frac{1}{2}$-approximation in expectation. But Yannakakis [34] pointed out that its derandomization via the method of conditional expectations yields Johnson's algorithm, which achieves a $\frac{2}{3}$-approximation.

In what follows we examine the performance of the deterministic 2-pass algorithm on the clause set

$$(x_1 \vee \overline{x}_3), (\overline{x}_1 \vee x_3), (x_1 \vee x_4), (\overline{x}_1 \vee \overline{x}_4), (x_2 \vee \overline{x}_4), (\overline{x}_2 \vee x_4), (x_2 \vee x_3), (\overline{x}_2 \vee \overline{x}_3)$$

and observe that fixing $x_1 = x_3 = \overline{x}_2 = \overline{x}_4 = 1$ satisfies all clauses.

Assume that the algorithm considers the variables in lexicographical order. When processing x_1, both Boolean assignments to x_1 satisfy two clauses, and no clause can be unsatisfied by the partial assignment that sets x_1 only. Thus, we have $t_1 = f_1 = \frac{2}{2}$, and the algorithm fixes $\sigma_1 = \frac{t_1}{t_1+f_1} = \frac{1}{2}$. The assignment $\sigma_2 = \frac{1}{2}$ follows analogously due to symmetry and by the fact that x_1 and x_2 have no common clauses.

Let us consider x_3: For $x_3 = 1$ two clauses are satisfied with probability one that were previously satisfied with probability $\frac{1}{2}$ only. Moreover, the two clauses containing literal $\overline{x}_3$ are unsatisfied with probability $\frac{1}{2}$ each. Thus, we have $t_3 = f_3 = 0$.

It turns out that in this case the analysis allows the algorithm to choose an arbitrary value in $[0, 1]$ as assignment probability for x_3. Assume in the following that the *best fixed tie-breaking strategy* for the fractional greedy algorithm assigns the assignment probability $\alpha \in [0, 1]$, whenever both the t- and the f-value are zero. By symmetry $t_4 = f_4 = 0$ holds for x_4 and hence the algorithm also chooses $\sigma_4 = \alpha$ $(= \sigma_3)$. Note that the clauses x_3 and x_4 appear in are symmetric (for the partial assignment to x_1 and x_2) if we obfuscate the names of the variables in these clauses; our bound on the approximation ratio holds for any tie-breaking strategy that assigns the same value to x_3 and x_4 in this case.

This concludes the first pass. Now the algorithm considers the variables once more in lexicographical order and picks the Boolean assignment for each variable that maximizes the respective conditional expectation as follows: For x_1 the expected satisfied weight is $1 + 1 + \alpha + (1 - \alpha) = 3$ if $x_1 = 1$, and the other variables are fixed independently according to the assignment probabilities given by their respective fractional value. This equals the expectation conditioned on $x_1 = 0$ and the respective random assignment for the other variables.

Again, assume that the best fixed tie-breaking scheme (in the above sense) for the second pass fixes $x_1 = \beta \in \{0, 1\}$. Due to symmetry, x_2 is also set to β. What does this mean for x_3 and x_4? We observe that clauses that are already satisfied by the Boolean assignments to x_1 and x_2 contribute the same amount to all conditional expectations in later rounds. Hence, if x_1 and x_2 are set to the same Boolean value, then the set of undecided clauses becomes $(x_3), (\overline{x}_3), (x_4), (\overline{x}_4)$. There is no assignment to x_3 and x_4 that satisfies more than two out of these four clauses. Summing up, the algorithm obtains six clauses, whereas the optimal (Boolean) assignment satisfies all eight. Thus, we have shown the following.

THEOREM 7.3. *The approximation ratio of the deterministic* 2*-pass algorithm for MAX SAT is exactly* $\frac{3}{4}$ *for any fixed tie-breaking scheme.*

8. Conclusions. We have given a simple randomized greedy algorithm that obtains a $\frac{3}{4}$-approximation. A natural question is whether randomness is inherently required by the greedy paradigm in order to achieve that performance ratio. To this end, we have utilized the model of priority algorithms to formalize a concept of greedy algorithms for MAX SAT. We have shown that none of these deterministic greedy algorithms can guarantee a $\frac{3}{4}$-approximation for our nonstandard data items. While to the best of our knowledge all greedy algorithms for MAX SAT fall within our definition and in particular can be implemented as priority algorithms with our nonstandard data items, it would be interesting to study adaptive priority algorithms for the strong data item that additionally reveals the signs of other variables appearing in joint clauses with the variable that we currently want to decide: either prove an inapproximability bound worse than $\frac{3}{4}$ (thereby improving the bound of Yung [36]) or show that this is impossible by giving such a $\frac{3}{4}$-approximation algorithm.

Moreover, we have presented a deterministic $\frac{3}{4}$-approximation algorithm that bypasses our inapproximability result by performing a second pass over the input. The same algorithmic idea can be applied to obtain deterministic $\frac{1}{2}$-approximation algorithms for MAX DICUT and MAX NAE-SAT that run in linear time; all these approximation ratios are tight. On the one hand, we wonder if the approach can be extended to obtain nontrivial approximations for constraint satisfaction problems. On the other hand, we are interested in the inherent limitations of deterministic algorithms that perform two passes of the input, but are only allowed to store very little information, e.g., a constant number of bits per variable.

Considering the performance of approximation algorithms for MAX SAT, only algorithms based on semidefinite programming achieve approximation ratios better than $\frac{3}{4}$. For MAX DICUT we have an analogous phenomenon for approximation ratio $\frac{1}{2}$. Are there efficient algorithms that beat the bound of $\frac{3}{4}$ (resp., $\frac{1}{2}$ for MAX DICUT) without solving an SDP?

Appendix A. A linear time implementation of greedy algorithms for MAX SAT.

We describe a linear time implementation for our approximation algorithms for MAX SAT (see also [30, 11]). The data structure we propose covers the randomized greedy algorithm studied in section 3, its variants in [28, 27], and Johnson's algorithm equipped with either an online or a random variable ordering [22, 13, 14, 28]. Moreover, our linear time implementation of the 2-pass algorithm of section 7 also relies on this data structure. Assuming the optimal LP solution is provided, we can also use it to implement the LP rounding algorithms of section 6 and [31] in linear time.

There are two main problems to address: When given a variable x_i, one needs to access all clauses that contain x_i and are not satisfied by S_{i-1}, the partial assignment to the first $i-1$ variables. In particular, the algorithms we consider require the weight and the number of unfixed literals for any such clause in order to decide x_i. Then, after fixing x_i, one has to "update" the clause set, i.e., remove satisfied and unsatisfied clauses to avoid the computational overhead of processing a clause that has been decided.

We assume that the formula is given as a collection of m clauses, where each clause C_j, $j \in [m]$, consists of a list of literals and has a nonnegative weight w_j. The size of the formula, i.e., its encoding length, is denoted by F. In particular, F is proportional to the total number of literals in the formula, counting multiple occurrences. In the following presentation we omit the space required to represent the clause weights, as they can be assumed to be in binary both in the input and in the data structure. We propose a data structure that yields a time complexity of $O(F)$.

Assume that the number of variables is n and their indices are $\{1, 2, \ldots, n\}$. For each literal x we create a doubly linked list L_x that will provide access to the clauses that are still undecided and contain x. Thus, there are two lists L_{x_i} and $L_{\overline{x}_i}$ for each variable x_i that can be accessed via an auxiliary array using the variable index. Now we perform a single run through the clause set: For clause C_j we create a clause object O_j that stores the weight w_j and the number of unfixed literals in C_j with respect to the current partial assignment. Further, for each literal $x \in C_j$ we append a new element E_j to L_x that contains a pointer to the clause object O_j, and store a pointer to E_j in O_j. The pointers in O_j can be kept in a simple linked list. We may charge the cost of setting up the data structure to the involved literals: The occurrence of literal x in clause C_j is charged for element E_j in L_x as well as the two pointers, and the first literal of C_j is charged for O_j. Thus, the data structure can be constructed in time $O(F)$.

We run the algorithm as follows. The algorithm specifies an ordering on the variables, for example the ordering given by the variable indices. If variable x_i is to be decided next, the algorithm cycles once through L_{x_i} and $L_{\overline{x}_i}$ to collect the information used to decide x_i. In the case of the algorithms presented in this paper this means computing t_i and f_i. Assume that the algorithm sets x_i to true (the case of false will be handled analogously). Then we remove the satisfied clauses from the formula by cycling through L_{x_i}: for each element $E_j \in L_{x_i}$ we access the corresponding clause object O_j and remove the clause from the L-lists of those literals that appear in C_j, using the pointers stored in the clause object O_j. Here we utilize that the L-lists are doubly linked, and hence the corresponding elements can be removed from the respective list in constant time. Then we delete the clause object O_j itself. Finally, we pass through $L_{\overline{x}_j}$ and decrement the number of unfixed literals for the corresponding clauses. If the counter drops to zero for some clause, we also remove it from the data structure by the procedure described above. Thus, at any time the data structure contains only undecided clauses. Observe that each literal x is fixed (at most) once by the algorithm; thus every occurrence of x is charged with constant cost.

Appendix B. A linear time implementation of the 2-pass algorithm.

We describe how the 2-pass algorithm can be implemented in linear time (measured in the input size) using the data structure presented in section A. That data structure allows efficient access to the clauses the current variable appears in.

In order to compute the assignment probabilities efficiently, we imagine that the algorithm would create a random assignment during its first pass. Accordingly, we define S_i to be a partial random assignment to $x_1, \dots, x_i$; in particular, S_n is a random assignment to all variables obtained by fixing each variable according to its respective assignment probability. Note that this assignment is never instantiated by the algorithm.

If we store for each clause the probability that it is unsatisfied under S_i, then each literal of the clause set is "charged" a constant number of arithmetic operations only. Here we use that the probability of a clause to be unsatisfied is just the product of the probabilities that each of its literals is fixed to zero. Moreover, we only need to consider and update the clauses that the current variable appears in. Thus, the number of updates to each clause is at most its number of literals. Therefore, the values t_i and f_i can be computed in amortized constant time.

Consider the second pass and assume that we are to decide x_i. Then the first $i-1$ variables have already been set to Boolean values: let T_{i-1} be the respective partial assignment. Let E_b for $b \in \{0, 1\}$ denote the expected weight of satisfied clauses if the variable x_i is fixed to b and the remaining unset variables (with respect to T_{i-1})

are decided according to their respective assignment probabilities.

The crucial observation is that we do not have to compute these conditional expectations E_0, E_1 explicitly to decide the assignment for variable x_i. In order to decide whether $E_1 \geq E_0$ holds, it is sufficient to consider the clauses x_i occurs in, since all other clauses contribute equally to both quantities. This allows us the compute the required quantities at amortized constant cost per literal in the input.

Acknowledgment. The authors would like to thank Ola Svensson for pointing out that the fractional greedy algorithm of Buchbinder et al. [11] can be turned into a deterministic $\frac{3}{4}$-approximation algorithm and for allowing them to present his idea in this article.

REFERENCES

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

[2] S. Angelopoulos and A. Borodin, *Randomized priority algorithms*, Theoret. Comput. Sci., 411 (2010), pp. 2542–2558.

[3] J. Argelich, C. M. Li, F. Manyà, and J. Planes, *MAX-SAT 2014: Ninth Max-SAT evaluation*, www.maxsat.udl.cat/14/, 2014, last accessed on 08/19/2015.

[4] P. Austrin, *Towards sharp inapproximability for any 2-CSP*, SIAM J. Comput., 39 (2010), pp. 2430–2463.

[5] A. Avidor, I. Berkovitch, and U. Zwick, *Improved approximation algorithms for MAX NAE-SAT and MAX SAT*, in Approximation and Online Algorithms, WAOA 2005, T. Erlebach and G. Persinao, eds., Springer, Berlin, 2005, pp. 27–40.

[6] Y. Azar, I. Gamzu, and R. Roth, *Submodular Max-SAT*, in Algorithms—ESA 2011, C. Demetrescu and M. M. Halldórsson, eds., Springer, Berlin, 2011, pp. 323–334.

[7] A. Belov, D. Diepold, M. J. H. Heule, and M. Järvisalo, *Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions*, University of Helsinki, 2014.

[8] A. Borodin, J. Boyar, K. S. Larsen, and N. Mirmohammadi, *Priority algorithms for graph optimization problems*, Theoret. Comput. Sci., 411 (2010), pp. 239–258.

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

[10] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz, *A tight linear time (1/2)-approximation for unconstrained submodular maximization*, in FOCS '12, Proceedings of the 2012 IEEE 53rd Ammual Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 2012, pp. 649–658.

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

[12] S. O. Chan, J. R. Lee, P. Raghavendra, and D. Steurer, *Approximate constraint satisfaction requires large LP relaxations*, J. ACM, 63 (2016), pp. 34:1–34:22.

[13] J. Chen, D. K. Friesen, and H. Zheng, *Tight bound on Johnson's algorithm for maximum satisfiability*, J. Comput. System Sci., 58 (1999), pp. 622–640.

[14] K. P. Costello, A. Shapira, and P. Tetali, *Randomized greedy: New variants of some classic approximation algorithms*, in Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2011, pp. 647–655.

[15] S. Davis and R. Impagliazzo, *Models of greedy algorithms for graph problems*, Algorithmica, 54 (2009), pp. 269–317.

[16] L. Engebretsen, *Simplified tight analysis of Johnson's algorithm*, Inform. Process. Lett., 92 (2004), pp. 207–210.

[17] U. Feige and M. X. Goemans, *Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT*, in Third Israel Symposium on the Theory of Computing and Systems, 1995, pp. 182–189.

[18] M. X. Goemans and D. P. Williamson, *New 3/4-approximation algorithms for the maximum satisfiability problem*, SIAM J. Discrete Math., 7 (1994), pp. 656–666.

[19] M. X. Goemans and D. P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. ACM, 42 (1995), pp. 1115–1145.

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

[21] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13–30.

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

[23] H. J. Karloff and U. Zwick, *A 7/8-approximation algorithm for MAX 3SAT?*, in Proceedings, 38th Annual Symposium on Foundations of Computer Science, 1997, pp. 406–415.

[24] J. M. Kleinberg and É. Tardos, *Algorithm Design*, Addison-Wesley, Harlow, UK, 2006.

[25] M. Lewin, D. Livnat, and U. Zwick, *Improved rounding techniques for the MAX 2-SAT and MAX DI-CUT problems*, in Integer Programming and Combinatorial Optimization, IPCO 2002, Springer, Berlin, 2002, pp. 67–82.

[26] S. Matuura and T. Matsui, *0.935-Approximation Randomized Algorithm for MAX–2SAT and Its Derandomization*, Technical Report METR 2001-03, Department of Mathematical Engineering and Physics, The University of Tokyo, Tokyo, Japan, 2001.

[27] M. Poloczek, *Bounds on greedy algorithms for MAX SAT*, in Algorithms—ESA 2011, C. Demetrescu and M. M. Halldórsson, eds., Springer, Berlin, 2011, pp. 37–48.

[28] M. Poloczek and G. Schnitger, *Randomized variants of Johnson's algorithm for MAX SAT*, in Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2011, pp. 656–663.

[29] M. Poloczek and D. P. Williamson, *An experimental evaluation of fast approximation algorithms for the maximum satisfiability problem*, in Experimental Algorithms, SEA 2016, A. Goldberg and A. Kulikov, eds., Springer, Cham, 2016, pp. 246–261.

[30] M. Poloczek, D. P. Williamson, and A. van Zuylen, *On some recent approximation algorithms for MAX SAT*, in LATIN 2014: Theoretical Informatics, A. Pardo and A. Biola, eds., Springer, Berlin, 2014, pp. 598–609.

[31] A. van Zuylen, *Simpler 3/4-approximation algorithms for MAX SAT*, in WAOA '11, Proceedings of the 9th International Conference on Approximation and Online Algorithms, Springer, Berlin, 2011, pp. 188–197.

[32] D. P. Williamson, *Lecture Notes in Approximation Algorithms, Fall 1998*, IBM Research Report RC 21409, IBM Research, 1999.

[33] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*, Cambridge University Press, Cambridge, UK, 2011.

[34] M. Yannakakis, *On the approximation of maximum satisfiability*, J. Algorithms, 17 (1994), pp. 475–502.

[35] A. Chi-Chih Yao, *Lower bounds by probabilistic arguments*, in 24th Annual Symposium on Foundations of Computer Science, IEEE, New York, 1983, pp. 420–428.

[36] C. K. Yung, *Inapproximation Result for Exact Max-2-SAT*, unpublished manuscript, 2011.

[37] U. Zwick, *Computer assisted proof of optimal approximability results*, in Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2002, pp. 496–505.