Streaming Algorithms for Maximum Satisfiability

Hoa T. Vu [*]

Abstract

We study the maximum satisfiability (Max-Sat) problem in the streaming setting. In this problem, we are given m clauses that are disjunctions of literals drawn from n boolean variables and the objective is to find a boolean assignment to the variables that maximizes the number of satisfied clauses. While Max-Sat is NP-Hard, it admits various polynomial time approximations. In the streaming model, input clauses are presented in an online fashion and the goal is to obtain a good approximation while using sublinear $o(mn)$ memory. We present randomized single-pass algorithms that w.h.p [1] yield

- a $1-\epsilon$ approximation using $\tilde{O}(n/\epsilon^3)$ space [2] and exponential time, and
- a $3/4-\epsilon$ approximation using $\tilde{O}(n/\epsilon^2)$ space and polynomial time.

For the Max-AND-Sat problem where clauses are conjunctions of literals, we show that any single-pass algorithm that approximates the optimum up to a factor better than 1 ± 0.5 with success probability at least $1-1/(m^2n^2)$ must use $\Omega(mn)$ space.

We also consider an interesting related problem called Max-Exactly1-Sat in which each clause is satisfied if and only if exactly one literal in it is true. For this problem, we present a single-pass, $\tilde{O}(n)$-space, polynomial time streaming algorithm that yields an $\Omega(1/\log n)$ approximation w.h.p.

Our streaming algorithms are based on simple meta algorithms that may also be implemented in other sublinear models.

[*]San Diego State University, hvu2@sdsu.edu

[1]W.h.p denotes "with high probability". Here, we consider $1-1/\operatorname{poly}(n,m)$ as high probability.

[2]$\tilde{O}$ hides polylog factors.

1 Introduction

Streaming Max-Sat. The (boolean) satisfiability problem is arguably one of the most famous problems in computer science since it is the first problem proven to be NP-Complete [Coo71, Tra84] (also known as Cook-Levin theorem). A satisfiability instance is a conjunction of m clauses $C_1 \wedge C_2 \wedge \ldots \wedge C_m$ where each clause C_j is a disjunction of literals drawn from a set of n boolean variables $x_1, \ldots, x_n$ (a literal is either a variable or its negation). Deciding whether such expression is satisfiable is NP-Complete and therefore we usually aim to maximize the number of satisfied clauses instead. This is known as the maximum satisfiability (Max-Sat) problem. But Max-Sat is NP-Hard even when each clause has at most two literals [GJS76]. To this end, researchers have developed approximation algorithms for this problem.

Boolean satisfiability also finds numerous applications in model-checking, design debugging, AI planning, bioinformatics, etc. For a comprehensive overview, see [MS08].

Max-Sat can be approximated up to a factor $3/4$ using linear programming (LP) [GW94], network flow [Yan94], or a careful greedy approach [PSWvZ17]. One can also obtain an approximation slightly better than 3/4 using semedefinite programming (SDP) [GW95, ABZ05].

In this paper, we study Max-Sat in the streaming model. Our streaming algorithms are based on simple meta algorithms that may also be implemented in other sublinear settings. There are two natural streaming models for this problem. In the *edge-arrival* model, each stream token is (x_i, C_j) or $(\overline{x_i}, C_j)$ which means that the literal x_i or $\overline{x_i}$ respectively is in the clause C_j. In the *clause-arrival* model, clauses are presented one by one in the stream. The objective is to use sublinear space $o(mn)$ while maintaining a guaranteed approximation. To facilitate the discussion, we only consider the clause-arrival model although our algorithms also work in the edge-arrival model with appropriate bookkeepings. However, the more important point is that it is often far more difficult to prove lower bounds in the clause-arrival model.

For simplicity, we assume m and n are provided. Furthermore, the memory use is measured in terms of $O(\max\{\log n, \log m\})$-bit words since we need $\Omega(\max\{\log n, \log m\})$ bits to encode IDs of variables and clauses. For Max-Sat, our main result are as follows.

Theorem 1.1 (Main result 1)**.** *Let $T = \min\{m, n/\epsilon^2\}$. There exists*

- *A $O(T \cdot \log m)$-space, polynomial time, single-pass streaming algorithm that yields a $3/4 - \epsilon$ approximation to* Max-Sat *w.h.p.*
- *A $O(T/\epsilon \cdot \log m)$-space algorithm, single-pass streaming algorithm that yields a $1 - \epsilon$ approximation to* Max-Sat *w.h.p.*

Throughout this paper, algorithms actually output an assignment to the variables along with an estimate for the number of satisfied clauses. On the other hand, lower bounds are only for approximating the optimum (without necessarily outputting the corresponding assignment). By adapting the lower bound for streaming Max-Cut (i.e., estimating the size of the largest cut in a graph) by Kapralov and Krachun [KK19], we show that approximating the optimum up to a factor better than $1 \pm 1/6$ requires $\Omega(n)$ space. This gives evidence that our algorithms are optimal or near-optimal (up to polylogarithmic factors).

Streaming Max-AND-Sat. In the maximum AND-satisfiability problem (Max-AND-Sat), each clause is a conjunction of literals (as opposed to being a disjunction of literals in Max-Sat). Our second main result is a space lower bound on any streaming algorithm that approximates the optimum to a factor better than 1 ± 0.5.

Theorem 1.2 (Main result 2)**.** *Assuming $n = \Omega(\log m)$, any single-pass streaming algorithm that approximates the optimum of* Max-AND-Sat *up to a factor better than 1 ± 0.5 with success probability at least $1 - 1/(m^2n^2)$ requires $\Omega(mn)$ space.*

Streaming Max-XOR-Sat and Max-Exactly1-Sat. If literals in a clauses are joint by XOR gates, the clause is satisfied if and only if the number of true literals modulo 2 is one (i.e., the number of true literals is odd). One may set each variable to true with probability 1/2 and obtain a 1/2 approximation (in expectation) to Max-XOR-Sat.

We consider a more interesting problem called Max-Exactly1-Sat where a clause is satisfied if and only if exactly one literal in it is true (this follows an alternative definition of "exclusive-or" for multiple variables). If every clause has at most two literals, then Max-Exactly1-Sat and Max-XOR-Sat are equivalent. We show that this problem seems to be harder than Max-XOR-Sat but easier than Max-AND-Sat.

Theorem 1.3 (Main result 3)**.** *There exists a single-pass, $\tilde{O}(n)$-space, polynomial time streaming algorithm that yields an $\Omega(1/\log n)$ approximation to* Max-Exactly1-Sat*.*

We can adapt the lower bound in [KK19] to show that approximating the optimum of Max-Exactly1-Sat and Max-XOR-Sat up to a factor better than 1 ± 0.5 requires $\Omega(n)$ space.

Related work. Approximation algorithms for Max-Sat using different techniques such as random assignment, network flow, linear programming, greedy, and semidefinite programming can be found in [Joh74, Yan94, GW94, PSWvZ17, GW95, ABZ05]. There are also better approximations for special cases such as Max-2Sat where each clause has exactly two literals [MM, LLZ02]. Hastad showed the inapproximability result that unless $P = NP$, there is no polynomial time algorithm that yields an approximation better than 21/22 to Max-2Sat [Hås01].

We note that a greedy algorithm in [PSWvZ17] is stated as a "2-pass algorithm". However, their algorithm requires going through each variable and (randomly) assigns it to true or false depending on both the number of satisfied and not-yet-unsatisfied clauses resulted by each choice. This is not implementable in the streaming setting unless one uses $\Omega(n)$ passes over the stream.

In the streaming model, Guruswami et al. [GVV17] studied the problem of approximating the optimum of the boolean Max-2CSP problem using logarithmic space.

Max-Sat and Max-AND-Sat were also studied in [Tre98] in the guise of parallel algorithms. Trevisan showed that there is a parallel algorithm that finds a $3/4 - \epsilon$ approximation to Max-Sat in $O(\mathrm{poly}(1/\epsilon, \log m))$ time using $\mathrm{poly}(n, m)$ processors [Tre98]. One of our sampling results can improve the number of processors to just $O(n/\epsilon^2)$.

Notation. We write $x_i \in C_j$ if the literal x_i appears in clause C_j. We also use $|C_j|$ to denote the number of literals in C_j. We often write OPT(P) to denote the optimal value of the problem P, and when the context is clear, we drop P and just write OPT.

2 Streaming Algorithms

2.1 Algorithms for maximum satisfiability

The framework. Without loss of generality, we may assume that there is no *trivially true* clause, i.e., clauses that contain both a variable and its negation and there are no duplicate literals in a clause. We show that if we sample $O(n/\epsilon^2)$ clauses uniformly at random and run a constant approximation algorithm for Max-Sat on the sampled clauses, we obtain roughly the same approximation. We derive this result from the folklore fact that OPT $\geq m/2$ and a Chernoff-Union bound style argument.

Lemma 2.1. *Suppose $m = \omega(n/\epsilon^2)$ and we run an α approximation algorithm (where $\alpha \geq 1/2$) on $O(n/\epsilon^2)$ clauses sampled uniformly at random. Then, we obtain an $\alpha - \epsilon$ approximation to* Max-Sat *on the original clauses w.h.p.*

Proof. We recall the folklore fact that OPT $\geq m/2$. If we set each variable to true with probability 1/2, then each clause is satisfied with probability at least $1/2$. By linearity of expectation, at least $m/2$ clauses are satisfied and therefore, there is an assignment that satisfies at least half of the clauses.

We sample Kn/ϵ^2 clauses uniformly at random for some sufficiently large constant K. Suppose that an assignment A satisfies m_A clauses. Let the number of sampled clauses that A satisfies be m'_A and let $p = Kn/(\epsilon^2 m)$. We observe that $\mathrm{E}\left[m'_A\right] = pm_A$. Since $p\,\mathrm{OPT} \geq pm_A$, by an application of Chernoff bound (see Appendix for a careful derivation), we have

$$\Pr\left(m'_A = pm_A \pm \epsilon p\,\mathrm{OPT}\right) \geq 1 - e^{-\epsilon^2 p\,\mathrm{OPT}/3} \geq 1 - e^{-\epsilon^2 (Kn/(m\epsilon^2))\,\mathrm{OPT}/3} \geq 1 - e^{-Kn/6}\,. \tag{1}$$

The last inequality follows from the fact that OPT $\geq m/2$. Finally, by taking a union bound over 2^n distinct assignments, we deduce that with probability at least $1 - e^{-\Omega(n)}$, for all assignments A, we have $m'_A = pm_A \pm \epsilon p\,\mathrm{OPT}$.

Suppose an assignment $\tilde{A}$ is an α approximation to Max-Sat on the sampled clauses where α is a constant. Let A^* be the optimal assignment on the original input clauses. From the above, w.h.p, we have

$$pm_{\tilde{A}} + \epsilon p \,\mathrm{OPT} \geq m'_{\tilde{A}} \geq \alpha m'_{A^*} \geq \alpha \,\mathrm{OPT}\, p(1-\epsilon) \ .$$

Therefore, $m_{\tilde{A}} \geq \alpha \,\mathrm{OPT}(1-\epsilon) - \epsilon \,\mathrm{OPT} = (\alpha - O(\epsilon)) \,\mathrm{OPT}$. □

The second observation to obtain sublinear memory algorithms for Max-Sat is that large clauses (i.e., clauses with many literals) are probabilistically easy to satisfy. We define β-large clauses as clauses that have at least β literals.

Lemma 2.2. *For* Max-Sat*, if each literal is set to* true *independently with probability at least γ, then for some sufficiently large constant K, all $(K \log m)/\gamma$-large clauses are satisfied w.h.p.*

Proof. The probability that each $((K \cdot \log m)/\gamma)$-large clause is not satisfied is at most $(1-\gamma)^{(K \log m)/\gamma} \leq e^{-K \log m} \leq 1/\operatorname{poly}(m)$ which implies that all such clauses are satisfied w.h.p by appealing to a union bound over at most m large clauses. □

We now state a simple *meta sublinear algorithm*, formalized as Algorithm 1, that can easily be implemented in several sublinear settings. We will present two possible post-processing algorithms and the corresponding β values for this meta algorithm.

```
1 if $m = \omega(n/\epsilon^2)$ then sample $O(n/\epsilon^2)$ clauses uniformly at random as input.
2 Collect (sampled) clauses that are not $\beta$-large where $\beta = O((\log m)/\gamma)$.
3 Post-processing: Run an $\alpha$ approximation for Max-Sat on the collected clauses
  where each each literal is true independently with probability at least $\gamma$.
```

Algorithm 1: A meta algorithm for sublinear Max-Sat

Post-processing algorithm 1: An exponential time $1-\epsilon$ approximation. Here, we set $\beta = (K \log m)/\epsilon$ for some sufficiently large constant K. Suppose in post-processing, we run the exact (brute-forte) algorithm on the collected clauses. We would like to apply Lemma 2.2, but the difficulty is that the exact algorithm is deterministic. Our trick is to randomly perturb the assignment given by the exact algorithm (or the exact assignment for short).

The random perturbation trick: If the exact algorithm sets $x_i = q \in \{\text{true}, \text{false}\}$, we set the new value $x'_i = q$ with probability $1-\epsilon$ (and $x'_i = \overline{q}$ with probability ϵ). Hence, each literal is true with probability at least ϵ.

We appeal to Lemma 2.2 with $\gamma = \epsilon$ to conclude that all β-large clauses are satisfied w.h.p. Let W be the set of the collected clauses that are satisfied by the exact algorithm. By linearity of expectation, the expected number of satisfied clauses in W after we randomly perturb the exact assignment is

$$\sum_{C \in W} \Pr\left(C \text{ is satisfied}\right) \geq \sum_{C \in W} (1-\epsilon) = (1-\epsilon)|W| \ .$$

The first inequality follows from the observation that at least one literal in C must be true in the exact assignment and therefore the probability that it remains true is $1-\epsilon$. It is then easy to see that we have a $1-\epsilon$ approximation in expectation. In particular, let L and S be the set of β-large and small clauses respectively. Furthermore, let OPT_L and OPT_S be the number of satisfied clauses in L and S respectively in the optimal assignment. Clearly, $|W| \geq \text{SOL}_S$ and $|L| \geq \text{OPT}_L$. Let SOL denote the number of clauses satisfied by the algorithm. We have

$$\begin{aligned} \mathrm{E}\left[\text{SOL}\right] \geq (1-\epsilon)|W| + \left(1 - \frac{1}{\operatorname{poly}(m)}\right)|L| &\geq (1-\epsilon)(\text{OPT}_S + \text{OPT}_L) \\ &\geq (1-\epsilon)\,\text{OPT}\ . \end{aligned}$$

To obtain the desired approximation w.h.p, we can repeat the random perturbation algorithm multiple times and return the best solution (on the collected clauses).

Lemma 2.3. *Suppose there is an algorithm that yields an $\alpha \geq 1/2$ approximation to* Max-Sat *in expectation. By repeating the algorithm $O((\log m)/\epsilon)$ times and choosing the best solution, we yield an $\alpha-\epsilon$ approximation w.h.p.*

Appealing to Lemma 2.3, we can repeat the random perturbation algorithm $O((\log m)/\epsilon)$ times to satisfy $(1-\epsilon)|W|$ collected clauses w.h.p. Recall that all $((K\log m)/\epsilon)$-large clauses are satisfied with probability at least $1 - 1/\operatorname{poly}(m)$. By the union bound, they are satisfied in all $O((\log m)/\epsilon)$ trials of the perturbation algorithm and therefore w.h.p,

$$\text{SOL} \geq (1-\epsilon)|W| + |L| \geq (1-\epsilon)(\text{OPT}_S + \text{OPT}_L) \geq (1-\epsilon)\,\text{OPT}\ .$$

Post-processing algorithm 2: A polynomial time $3/4-\epsilon$ approximation. We can set $\beta = O(\log m)$ if we settle for a $3/4-\epsilon$ approximation. This saves a factor $1/\epsilon$ in the memory use. Consider the standard linear programming (LP) formulation for Max-Sat given by Goemans and Williamson [GW94].

$$\begin{aligned} (LP)\ \ \text{maximize}\ \ & \sum_{j=1}^{m} z_j \\ \text{subject to}\ \ & \sum_{i\in P_j} y_i + \sum_{i\in N_j}(1-y_i) \geq z_j && \text{for all } 1\leq j\leq m \\ & 0 \leq y_i, z_j \leq 1 && \text{for all } 1\leq i\leq n, 1\leq j\leq m\ . \end{aligned}$$

The integer linear program where $y_i, z_j \in \{0,1\}$ corresponds exactly to the Max-Sat problem. In particular, if x_i is in clause C_j then $i \in P_j$ and if $\overline{x_i}$ is in C_j then $i \in N_j$. We associate $y_i = 1$ with x_i being set to true and 0 if it is set to false. Similarly, we associate $z_i = 1$ with clause C_j being satisfied and 0 otherwise. Let $\text{OPT}(LP)$ denote the optimum of the above LP.

Lemma 2.4 ([GW94], Theorem 5.3)**.** *The optimal fractional solution of the linear program for* Max-Sat *can be rounded to yield an assignment that satisfies* $3/4\cdot\text{OPT}(LP) \geq 3/4\cdot\text{OPT}$

clauses in expectation. In this rounding algorithm, each variable is independently set to true *with probability $1/4 + y_i^*/2$ where y_i^* is the value of y_i in the optimal fractional solution of the linear program.*

Note that in the above randomized rounding, each literal is true with probability at least 1/4. Hence, we only need to collect clauses that are not $O(\log m)$-large and then find a $3/4$ approximation (in expectation) on the collected clauses by solving the LP and applying the rounding algorithm in Lemma 2.4. By repeating the randomized rounding $O((\log m)/\epsilon)$ times and taking the best assignment, we satisfy $3/4 \cdot |W|$ clauses among the collected clauses w.h.p. Furthermore, we can apply Lemma 2.2 (with $\gamma = 1/4$) to argue that all $O(\log m)$-large clauses are satisfied w.h.p. Therefore, w.h.p,

$$\text{SOL} \geq (3/4 - \epsilon)|W| + |L| \geq (3/4 - \epsilon)(\text{OPT}_S + \text{OPT}_L) \geq (3/4 - \epsilon)\,\text{OPT} \;.$$

Implementation in the streaming model. We finalize the proof of the first main result by outlining the implementation of the algorithms above in the streaming model.

Proof of Theorem 1.1. Recall that if $m = \omega(n/\epsilon^2)$, we can sample $O(n/\epsilon^2)$ clauses in the stream uniformly at random using Reservoir sampling [Vit85]. Therefore, we collect at most $T = \min\{m, n/\epsilon^2\}$ clauses. Among the sampled clauses, we simply stop storing (and ignore) a clause when the number of literals in it exceeds β. Hence, we always use $O(T\beta)$ space. We can run post-processing algorithm 1 where $\beta = (K \log m)/\epsilon$ and yield a $1 - \epsilon$ approximation. Alternatively, we set $\beta = (K \log m)$ and run the post-processing algorithm 2 to yield a $3/4 - \epsilon$ approximation.

□

Remark. It is possible to have an approximation better than $3/4 - \epsilon$ in polynomial post-processing time using semidefinite programming (SDP) instead of linear programming [GW95]. In the SDP-based algorithm that obtains a $0.7584 - \epsilon$ approximation, we also have the property that each literal is true with probability at least some constant. Hence, the analysis is analogous to what we outlined above. If we are satisfied with an approximation only in expectation, we can also drop the $\log m$ factor in β.

2.2 Algorithm for Max-Exactly1-Sat

We may make the following assumption that there is no *trivially false clause* in the input (i,e., clauses that contain at least two variables and their negations) because in clause-arrival streams, we see a clause all at once and are able to discard a clause if it is trivially false. We present a simple $\Omega(1/\log n)$ approximation algorithm to Max-Exactly1-Sat that can also be implemented in the streaming model (see Algorithm 2). The basic idea is simple: for each value $k \in [l]$, where $l = O(\log n)$, we set each variable to true independently with probability $p_k = 1/2^k$. We return the best solution among these attempts. Let SOL_k be the number of clauses satisfied by independently setting each variable to true with probability p_k and let $\text{SOL} = \max_k \text{SOL}_k$.

```
1 Let $l = O(\log n)$. For each $k \in [l]$, let $p_k = 1/2^k$. Let $\mathrm{SOL}_k$ be the number of
    clauses satisfied by setting each variable to true independently with probability $p_k$.
2 Return the assignment that corresponds to $\max_{k\in[l]} \mathrm{SOL}_k$.
```

Algorithm 2: Meta algorithm for Max-Exactly1-Sat

Lemma 2.5. *We have* $\mathrm{E}[\mathrm{SOL}] \geq \Omega(\mathrm{OPT}/\log n)$.

Proof. For the analysis, we partition clauses into three classes

$$\mathcal{C}^{(1)} = \{C_j : |C_j| = 1\},\ \mathcal{C}^{(2)} = \{C_j : C_j \text{ has both literals } x_i \text{ and } \overline{x_i} \text{ for some } x_i\},\ \text{and}$$
$$\mathcal{C}^{(3)} = \{C_j : C_j \notin \mathcal{C}^{(1)} \cup \mathcal{C}^{(2)}\}\ .$$

Obviously, $\max\{|\mathcal{C}^{(1)}|, |\mathcal{C}^{(2)}|, |\mathcal{C}^{(3)}|\} \geq \mathrm{OPT}/3$ and therefore, it suffices to design an $\Omega(1/\log n)$ approximation for each class.

Consider the algorithm that sets each variable to true independently with probability 1/2. Clearly, it satisfies at least $|\mathcal{C}^{(1)}|/2$ clauses in expectation.

Now, consider clauses in $\mathcal{C}^{(2)}$ and $\mathcal{C}^{(3)}$. We further divide these clauses into $l = O(\log n)$ sub-classes $\mathcal{C}_1^{(2)}, \ldots, \mathcal{C}_l^{(2)}$ and $\mathcal{C}_1^{(3)}, \ldots, \mathcal{C}_l^{(3)}$ where

$$\mathcal{C}_k^{(2)} = \{C_j \in \mathcal{C}^{(2)} : 2^k \leq |C_j| \leq 2^{k+1}\},\ \text{and}\ \mathcal{C}_k^{(3)} = \{C_j \in \mathcal{C}^{(3)} : 2^k \leq |C_j| \leq 2^{k+1}\}\ .$$

There must be some $\mathcal{C}_t^{(3)}$ such that $|\mathcal{C}_t^{(3)}| \geq |\mathcal{C}^{(3)}|/l = \Omega(|\mathcal{C}^{(3)}|/\log n)$. Consider the algorithm that sets each variable to true with probability $1/2^t$ independently. The probability that a size-r clause $C \in \mathcal{C}_t^{(3)}$ is satisfied is

$$rp_t(1-p_t)^{r-1} \geq \frac{r}{2^t}\left(1-\frac{1}{2^t}\right)^r \geq \left(1-\frac{1}{2^t}\right)^{2^{t+1}} \geq 1/e^3\ .$$

By linearity of expectation, we satisfy $|\mathcal{C}_t^{(3)}|/e^3 = \Omega(|\mathcal{C}^{(3)}|/\log n)$ clauses.

Similarly, there must be some $\mathcal{C}_y^{(3)}$ such that $|\mathcal{C}_y^{(2)}| = \Omega(|\mathcal{C}^{(2)}|/\log n)$. The probability that a size-r clause $C \in \mathcal{C}_y^{(2)}$ is satisfied is

$$(1-p_y)^{r-2} \geq \left(1-\frac{1}{2^y}\right)^{2^{y+1}} \geq 1/e^3\ .$$

As a result, we satisfy at least $|\mathcal{C}_y^{(2)}|/e^3 = \Omega(|\mathcal{C}^{(2)}|/\log n)$ clauses in expectation. □

We can repeat the above algorithm $O(\mathrm{polylog}(m,n))$ times and return the best solution to amplify the success probability.

Lemma 2.6. *Repeating Algorithm 2 at most* $O(\log^2 n \log m)$ *times and returning the best solution yields an* $\Omega(1/\log n)$ *approximation to* Max-Exactly1-Sat *w.h.p.*

We outline the implementation in the streaming model to conclude the next main result.

Proof of Theorem 1.3. It is straightforward to implement the above algorithm in the streaming model. We simply need to set each variable to true with probability p_k independently for each $k \in [l]$ and then count the number of satisfied clauses during the stream. We can run $O(\text{polylog}(n, m))$ copies of the aforementioned algorithm in parallel and return the best solution to obtain a $\Omega(1/\log n)$ approximation w.h.p according to Lemma 2.6. The algorithm uses $O(n \,\text{polylog}(m, n))$ space overall. $\square$

3 Lower Bounds

In this section, we provide space lower bounds for streaming algorithms for Max-Sat, Max-XOR-Sat, Max-Exactly1-Sat, and Max-AND-Sat. Recall that these lower bounds hold even if we just want to approximate the optimum OPT (without outputting the corresponding variables assignment).

Lower bounds for Max-Sat, Max-XOR-Sat, and Max-Exactly1-Sat. The streaming maximum cut problem asks to output an approximation of the size of the maximum cut OPT(Max-Cut) $= \max_{S \subset V} |E(S, V \setminus S)|$ of a graph $G = (V, E)$ (with n vertices and m edges). In this problem, the stream consists of edges of the underlying graph. One can use Max-Sat, Max-XOR-Sat, or Max-Exactly1-Sat to encode the maximum cut problem. Consider the cut between S and $V \setminus S$. We set $x_u =$ true if and only if $u \in S$. When an edge uv arrives in the graph stream, we put $(x_u \oplus x_v)$ in the Max-XOR-Sat stream ($\oplus$ denotes the XOR function) and two clauses $(x_u \vee x_v)$ and $(\overline{x_u} \vee \overline{x_v})$ in the Max-Sat stream. Note that there are n variables corresponding to vertices in the graph.

Both clauses $(x_u \vee x_v)$ and $(\overline{x_u} \vee \overline{x_v})$ are satisfied if and only if exactly one of x_u and x_v is in S; otherwise, exactly one of those two clauses is satisfied. We also note that $|E| \leq 2\,\text{OPT(Max-Cut)}$. It is easy to see that

$$\begin{aligned} \text{OPT(Max-XOR-Sat)} &= \text{OPT(Max-Cut)}, \text{ and} \\ \text{OPT(Max-Sat)} &= |E| + \text{OPT(Max-Cut)} \leq 3\,\text{OPT(Max-Cut)}\,. \end{aligned}$$

Hence, an α approximation to OPT(Max-XOR-Sat) yields an α approximation to OPT(Max-Cut). Furthermore, suppose we could approximate OPT(Max-Sat) up to a factor $(1 \pm (1/6 - \epsilon))$ then

$$\begin{aligned} (1 \pm (1/6 - \epsilon))\,\text{OPT(Max-Sat)} - |E| &= \text{OPT(Max-Sat)} - |E| \pm (1/6 - \epsilon)\,\text{OPT(Max-Sat)} \\ &= \text{OPT(Max-Cut)} \pm (0.5 - 3\epsilon)\,\text{OPT(Max-Cut)} \end{aligned}$$

which implies we can approximate OPT(Max-Cut) up to a factor better than 1 ± 0.5 since we count $|E|$ exactly. Kapralov and Krachun showed that approximating OPT(Max-Cut) up to a factor better than 1 ± 0.5 requires $\Omega(n)$ space [KK19]. We remark that their lower bound holds for any any algorithm that finds an estimate within the range from OPT(Max-Cut)$/2$ to $2\,$OPT(Max-Cut) and that $(1 \pm (0.5 - 3\epsilon))\,$OPT(Max-Cut) lies within that range. Since we only use 2 literals in each clause, Max-XOR-Sat and Max-Exactly1-Sat are the same. We rephrase their lower bound in terms of Max-Sat and Max-Exactly1-Sat below.

Theorem 3.1 ([KK19]). *Any single-pass streaming algorithm that approximates* $\mathrm{OPT}(\text{Max-Sat})$ *up to a factor better than* $1 \pm 1/6$*, or approximates* $\mathrm{OPT}(\text{Max-Exactly1-Sat})$, $\mathrm{OPT}(\text{Max-XOR-Sat})$ *up to a factor better than* 1 ± 0.5 *must use* $\Omega(n)$ *space.*

Lower bound for Max-AND-Sat. Our last main result is to show that Max-AND-Sat does not admit a sublinear space streaming algorithm that approximates OPT up to a factor better than 1 ± 0.5. In our lower bound, $n = \Omega(\log m)$.

Consider the following one-way communication problem. Alice receives a length-nm random bitstring A that Bob wants to learn about say with success probability at least $1 - 1/\operatorname{poly}(n, m)$. Alice must send a message of length $\Omega(mn)$ to Bob. Otherwise, Bob can distinguish 2^{nm} different inputs using $o(nm)$ bits of communication w.h.p which is a contradiction.

We can index A as $[m] \times [n]$. We may also assume that Alice and Bob share public unbiased random bits Z_{ji} for $j \in [m]$ and $i \in [n]$. According to Newman's theorem [New91], we can translate a protocol using public randomness with success probability at least $1 - 1/\operatorname{poly}(m, n)$ to a protocol using private randomness with success probability at least $1 - 2/\operatorname{poly}(m, n)$ using only $O(\log(nm))$ extra bits of communication.

Suppose that there is an algorithm $\mathcal{A}$ that can approximate OPT up to a factor better than (1 ± 0.5) with probability at least $1 - 1/m^2n^2$. We note that approximating OPT up to a factor better than (1 ± 0.5) distinguishes the cases $\mathrm{OPT} = 2$ and $\mathrm{OPT} = 1$.

If $A_{ji} = 1$, then, using the public unbiased bit Z_{ji}, Alice adds the literal x_i to clause C_j if $Z_{ji} = 1$ and adds the literal $\overline{x_i}$ to clause C_j if $Z_{ji} = 0$. Otherwise if $A_{ji} = 0$, then C_j contains neither x_i nor $\overline{x_i}$. She then puts $C_1, \ldots, C_m$ in the stream. The lemma below shows that it is not possible to satisfy more than one clause among $C_1, \ldots, C_m$ w.h.p.

Lemma 3.1. *Suppose* $n = \Omega(\log m)$*. With probability at least* $1 - e^{-\Omega(n)}$*, at most one clause among* $C_1, \ldots, C_m$ *can be satisfied.*

Proof. Consider any two clauses C_j and C_k that Alice puts in the stream. C_i and C_k can be both satisfied if and only if there does not exist $i \in [n]$ such that $(x_i \in C_j \wedge \overline{x_i} \in C_k)$ or $(\overline{x_i} \in C_j \wedge x_i \in C_k)$. For any fixed $i \in [n]$, we have $\Pr\left(x_i \in C_j \wedge \overline{x_i} \in C_k\right) = 1/16$. The probability that there does not exist $i \in [n]$ such that $x_i \in C_j$ and $\overline{x_i} \in C_k$ is $(1 - 1/16)^n = (15/16)^n$. Therefore, with probability at most $(15/16)^n$, C_j and C_k can be both satisfied. Appealing to a union bound over $\binom{m}{2}$ pairs of clauses, the probability that at most one clause among $C_1, \ldots, C_m$ can be satisfied is at least

$$1 - \binom{m}{2}(15/16)^n \geq 1 - e^{-\Omega(n)}$$

where we used the assumption that $m = \Omega(m)$ to yield the above inequality. $\square$

Alice then sends the memory of the streaming algorithm to Bob. Recall that Bob wants to recover the bitstring A. He recovers each junk $A_{j1}, \dots, A_{jn}$ for $j \in [m]$ one by one. For a fixed $j \in [m]$, Bob makes a clause $C_{m+1}^{(j)}$ in which $x_i \in C_{m+1}$ if $Z_{ji} = 1$ and $\overline{x_i} \in C_{m+1}$ if $Z_{ji} = 0$.

The *key observation* is that initially C_j and $C_{m+1}^{(j)}$ can be both satisfied since if a variable appears in both clauses, it appears the same way (negated or unnegated); hence, if Bob puts $C_{m+1}^{(j)}$ in the stream, OPT $= 2$. For $i = 1, 2, \dots, n$, Bob changes the literal x_i to $\overline{x_i}$ (or $\overline{x_i}$ to x_i) in $C_{m+1}^{(j)}$ (we say Bob *flips* x_i) and puts the modified $C_{m+1}^{(j)}$ in the stream instead of the original $C_{m+1}^{(j)}$. He then checks if OPT $= 1$ using the streaming algorithm $\mathcal{A}$ and learns whether $A_{ji} = 1$. He then unflips x_i and proceeds to learn to the next bit $A_{j,i+1}$ in the same manner. This procedure is possible since Bob has the memory state of the algorithm on the stream $C_1, \dots, C_m$ from Alice. More formally, the procedure in Algorithm 3 allows Bob to recover $A_{j1}, \dots, A_{jn}$.

```
1 for $i = 1, 2, \dots, n$ do
2     if $x_i$ is in $C_{m+1}^{(j)}$ then change $x_i$ to $\overline{x_i}$ in $C_{m+1}^{(j)}$
3     else change $\overline{x_i}$ to $x_i$ in $C_{m+1}^{(j)}$
4     Use $\mathcal{A}$ to check if OPT $= 1$ on the stream $C_1, \dots, C_m, C_{m+1}^{(j)}$
5     if OPT $= 1$ then $A_{ji} = 1$
6     else $A_{ji} = 0$
7     Undo the change to $x_i$ in $C_{m+1}^{(j)}$
```

Algorithm 3: Procedure to recover $A_{j1}, \dots, A_{jn}$

We argue that Bob correctly recovers A_{ji}. If $A_{ji} = 0$, then both x_i and $\overline{x_i}$ are not in C_j and therefore if Bob flips x_i in $C_{m+1}^{(j)}$, both clauses C_j and $C_{m+1}^{(j)}$ can still be both satisfied. In this scenario, OPT $= 2$.

If $A_{ji} = 1$, then either x_i or $\overline{x_i}$ is in C_j. If Bob flips x_i, only one clause among C_j and $C_{m+1}^{(j)}$ can now be satisfied since x_i appears negated in one clause and unnegated in the other. Therefore, Bob will know that $A_{ji} = 1$ if OPT $= 1$.

Hence, Bob can recover A_{ji} for each $i \in [n]$ and $j \in [m]$. The final hurdle is to show that, w.h.p, no other C_l (where $l \neq j$) can be satisfied along with $C_{m+1}^{(j)}$ during these flips. The proof is almost identical to the proof of Lemma 3.1.

Lemma 3.2. *Suppose $n = \Omega(\log m)$. For a fixed $j \in [m]$, when Bob flips x_i in Algorithm 3 to recover A_{ji}, with probability at least $1 - e^{-\Omega(n)}$, for all $l \in [m] \setminus \{j\}$, clauses C_l and $C_{m+1}^{(j)}$ cannot be both satisfied.*

Proof. When Bob flips x_i, the sets of literals in $C_{m+1}^{(j)}$ and C_l are still random and independent of each other. Each literal independently appears in $C_{m+1}^{(j)}$ with probability 1/2 and independently appears in C_l with probability 1/4. For any fixed $k \in [n], l \in [m] \setminus \{j\}$, we now have $\Pr\left(x_k \in C^{(j)}_{m+1} \wedge \overline{x_k} \in C_l\right) = 1/8$. Thus, the probability that there does not exist $k \in [n]$ such that $x_k \in C^{(j)}_{m+1}$ and $\overline{x_k} \in C_l$ is $(1 - 1/8)^n = (7/8)^n$. Taking a union bound over $l \neq j$ and using the assumption that $n = \Omega(\log m)$, we deduce that no other clause C_l (except C_j) can be satisfied simultaneously with $C^{(j)}_{m+1}$ during these flips with probability at least $1 - e^{-\Omega(n)}$. $\square$

Since the algorithm failure probability is at most $1/(n^2m^2)$ and there are nm flips to recover all the bits of A, the overall failure probability of the recover procedure is at most

$$1/(nm) + (nm) \cdot e^{-\Omega(n)} + e^{-\Omega(n)} \leq 1/\operatorname{poly}(n, m) .$$

If the streaming algorithm $\mathcal{A}$ uses $o(nm)$ space, Bob can distinguish 2^{nm} different inputs using $o(nm)$ bits of communication w.h.p which is a contradiction. Therefore, we have proved the lower bound in Theorem 1.2.

References

[ABZ05] Adi Avidor, Ido Berkovitch, and Uri Zwick. Improved approximation algorithms for MAX NAE-SAT and MAX SAT. In *WAOA*, volume 3879 of *Lecture Notes in Computer Science*, pages 27–40. Springer, 2005.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158. ACM, 1971.

[GJS76] M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified np-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.

[GVV17] Venkatesan Guruswami, Ameya Velingker, and Santhoshini Velusamy. Streaming complexity of approximating max 2csp and max acyclic subgraph. In *APPROX-RANDOM*, volume 81 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[GW94] Michel X. Goemans and David P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM J. Discrete Math.*, 7(4):656–666, 1994.

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.

[Hås01] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

[Joh74] David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974.

[KK19] Michael Kapralov and Dmitry Krachun. An optimal space lower bound for approximating MAX-CUT. In *STOC*, pages 277–288. ACM, 2019.

[LLZ02] Michael Lewin, Dror Livnat, and Uri Zwick. Improved rounding techniques for the MAX 2-sat and MAX DI-CUT problems. In *IPCO*, volume 2337 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2002.

[MM] Shiro Matuura and Tomomi Matsui. 0.935-approximation randomized algorithm for max 2sat and its derandomization.

[MS08] Joao Marques-Silva. Practical applications of boolean satisfiability. In *2008 9th International Workshop on Discrete Event Systems*, pages 74–80. IEEE, 2008.

[New91] Ilan Newman. Private vs. common random bits in communication complexity. *Inf. Process. Lett.*, 39(2):67–71, 1991.

[PSWvZ17] Matthias Poloczek, Georg Schnitger, David P. Williamson, and Anke van Zuylen. Greedy algorithms for the maximum satisfiability problem: Simple algorithms and inapproximability bounds. *SIAM J. Comput.*, 46(3):1029–1061, 2017.

[Tra84] Boris A. Trakhtenbrot. A survey of russian approaches to perebor (brute-force searches) algorithms. *IEEE Annals of the History of Computing*, 6(4):384–400, 1984.

[Tre98] Luca Trevisan. Parallel approximation algorithms by positive linear programming. *Algorithmica*, 21(1):72–88, 1998.

[Vit85] Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[Yan94] Mihalis Yannakakis. On the approximation of maximum satisfiability. *J. Algorithms*, 17(3):475–502, 1994.

A Omitted Proofs

A more careful derivation of Eq. 1

Theorem A.1 (Chernoff bound). *If $X = \sum_{i=1}^{n} X_i$ where X_i are negatively-correlated binary random variables and $\Pr(X_i = 1) = p$, then for all $\eta > 0$:*

$$\Pr\left(|X - pn| \geq \eta pn\right) \leq \exp\left(-\frac{\eta^2}{2+\eta} pn\right) .$$

Let $m_A = y$. Without loss of generality, suppose the assignment A satisfies clauses $C_1, \dots, C_y$. We define the indicator variable $X_i = [C_i \text{ is sampled}]$ and so $m'_A = \sum_{i=1}^{y} X_i$. Let $\eta = \epsilon \operatorname{OPT}/y$. We have

$$\begin{aligned}
\Pr\left(|m'_A - py| \geq \epsilon p \operatorname{OPT}\right) &\leq \exp\left(-\frac{\eta^2}{2+\eta} py\right) \\
&\leq \exp\left(-\frac{\epsilon^2 \operatorname{OPT}^2/y^2}{2+\epsilon \operatorname{OPT}/y} py\right) \\
&\leq \exp\left(-\frac{\epsilon^2 \operatorname{OPT}^2}{2y+\epsilon \operatorname{OPT}} p\right) \\
&\leq \exp\left(-\epsilon^2 \operatorname{OPT} p/3\right) \ .
\end{aligned}$$

The last inequality is because $3 \operatorname{OPT} \geq 2y + \epsilon \operatorname{OPT}$.

Proof of Lemma 2.3. We have $\operatorname{E}[m - \text{SOL}] \leq m - \alpha \operatorname{OPT}$. By Markov inequality,

$$\begin{aligned}
\Pr(\text{SOL} \leq (1-\epsilon)\alpha \operatorname{OPT}) &= \Pr(m - \text{SOL} \geq m - (1-\epsilon)\alpha \operatorname{OPT}) \\
&\leq \frac{m - \alpha \operatorname{OPT}}{m - (1-\epsilon)\alpha \operatorname{OPT}} \\
&= \frac{1}{1 + \epsilon\alpha \operatorname{OPT}/(m - \alpha \operatorname{OPT})} \ .
\end{aligned}$$

Because $m/4 \leq \alpha \operatorname{OPT}$, we have $\alpha \operatorname{OPT}/(m - \alpha \operatorname{OPT}) \geq 1/3$. Thus,

$$\Pr(\text{SOL} \leq (1-\epsilon)\alpha \operatorname{OPT}) \leq \frac{1}{1+\epsilon/3}$$

Hence, by repeating the algorithm $O(\log_{1+\epsilon/3} m) = O(1/\epsilon \cdot \log m)$ times and choosing the best solution, we obtain an $\alpha - \epsilon$ approximation w.h.p.

□

Proof of Lemma 2.6. Let m be the number of clauses after ignoring trivially false clauses. Let $\alpha = \Omega(1/\log n)$ be the expected approximation of the algorithm. We have $\operatorname{E}[m - \text{SOL}] \leq m - \alpha \operatorname{OPT}$. By Markov inequality,

$$\begin{aligned}
\Pr\left(\text{SOL} \leq \frac{\alpha \operatorname{OPT}}{2}\right) &= \Pr\left(m - \text{SOL} \geq m - \alpha\frac{\operatorname{OPT}}{2}\right) \\
&\leq \frac{m - \alpha \operatorname{OPT}}{m - \alpha \operatorname{OPT}/2} \\
&= \frac{m - \alpha \operatorname{OPT}}{m - \alpha \operatorname{OPT} + \alpha \text{OPT}/2} \\
&= \frac{1}{1 + \alpha \text{OPT}/(2(m - \alpha \text{OPT}))} \ .
\end{aligned}$$

We argued that $\alpha\,\text{OPT} \geq K\alpha^2 m$ for some constant K. Thus,

$$\frac{1}{2(m - \alpha\,\text{OPT})} \geq \frac{1}{2(m - K\alpha^2 m)}$$

Therefore,

$$\begin{aligned}\Pr\left(\text{SOL} \leq \frac{\alpha\,\text{OPT}}{2}\right) \leq \left(1 + \frac{K\alpha^2 m}{2(m - K\alpha^2 m)}\right)^{-1} &= \left(1 + \frac{K\alpha^2}{2(1 - K\alpha^2)}\right)^{-1} \\ &\leq \frac{1}{1 + K\alpha^2/2}\,.\end{aligned}$$

By repeating the algorithm $O(\log_{1+K\alpha^2/2} m) = O(\log^2 n \log m)$ times and choosing the best solution, we obtain an $\alpha - \epsilon$ approximation w.h.p. $\square$