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 non-trivial 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;
- A $3/4-\epsilon$ approximation using $\tilde{O}(n/\epsilon^2)$ space and polynomial time. If there are no duplicate clauses, the space can be further improved to $\tilde{O}(n/\epsilon)$.

We complement these algorithms with a lower bound showing that any single-pass algorithm that approximates the optimum of `MAX-SAT` up to a factor better than $3/4$ requires $\Omega(n)$ space.

On the other hand, 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 $1/2$ with success probability at least $1-1/(mn)^2$ must use $\Omega(mn)$ space.

*San Diego State University, hvu2@sdsu.edu

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

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

1 Introduction

Problem overview. 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]. This is also known as Cook-Levin theorem. A satisfiability instance is a conjunction of m clauses $C_1, C_2, \ldots, C_m$ 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 an expression is satisfiable is NP-Complete and therefore we often aim to find an assignment to the variables that approximately maximizes the number of satisfied clauses. This is known as the maximum satisfiability (`MAX-SAT`) problem. This problem is still NP-Hard even when each clause has at most two literals [GJS76].

`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. In the *edge-arrival* model, each stream token is in the form (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 obtaining a guaranteed non-trivial 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.

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`).

Motivation. Maximum satisfiability has found numerous applications in model-checking, software package management, design debugging, AI planning, bioinformatics, diagnosis, etc. [JSS00, ABL+10, CSMV10, AM07, HPJ+17, Cus08, LM06, MJIM15]. For a comprehensive overview of applications, see [MS08]. Many of these applications require solving large instances of `MAX-SAT`. For example, balancing tradeoffs in software analysis requires solving `MAX-SAT` instances with an extremely large number of clauses [SZGN17]. Other `MAX-SAT` benchmarks that arise from real-life applications also have large sizes [3]. This motivates us to study this problem in the streaming setting that aims to use sublinear memory. Such algorithms are helpful since many `MAX-SAT` instances are too large to fit in a computer's main memory. Furthermore, our algorithms also produce a compressed version of the original input that preserves approximations. This may speed up various algorithms and heuristics.

Our results. For simplicity, we assume m and n are provided. Furthermore, the memory use is measured in terms of $O(\max\{\lg n, \lg m\})$-bit words since we need at least $\Omega(\max\{\lg n, \lg m\})$ bits to encode IDs of the variables and clauses. For `MAX-SAT`, we show that it is possible to obtain a non-trivial approximation using only $O(n \cdot \text{poly}(\lg n, 1/\epsilon))$ space.

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

- *A Monte Carlo, $O(T \cdot \lg m)$-space, polynomial-time, and single-pass streaming algorithm that yields a $3/4 - \epsilon$ approximation to* `MAX-SAT` *w.h.p.*
- *A Monte Carlo, $O(T/\epsilon \cdot \lg m)$-space algorithm, single-pass streaming algorithm that yields a $1 - \epsilon$ approximation to* `MAX-SAT` *w.h.p.*

[3] Example benchmarks can be found at `https://maxsat-evaluations.github.io/2019/benchmarks.html` .

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 an approximation better than 3/4 requires $\Omega(n)$ space (Theorem 11). This gives evidence that our algorithms are optimal or near-optimal (up to polylogarithmic factors). The algorithms in Theorem 1 are based on the following key observations.

Observation 1: If $m = \omega(n/\epsilon^2)$ and we sample $\Theta(n/\epsilon^2)$ clauses uniformly at random, then w.h.p. an α approximation on the sampled clauses corresponds to an $\alpha - \epsilon$ approximation on the original input (Lemma 4).

Observation 2: If a clause is large, it can be satisfied w.h.p. as long as each literal is set to `true` independently with a not-too-small probability and therefore we may ignore such clauses (Lemma 5).

Based on the above observations, we proceed by first ignoring large clauses. Then, among the remaining (small) clauses, we sample $\Theta(n/\epsilon^2)$ clauses uniformly at random, denoted by W. Finally, we run some randomized α approximation algorithm on W in post-processing in which every literal is set to `true` with some small probability. This will lead to an $\alpha - \epsilon$ approximation on the original set of clauses w.h.p.

In the $3/4 - \epsilon$ approximation in Theorem 1, the dependence on ϵ is $1/\epsilon^2$ which may be prohibitive for some applications. We show that it is possible to improve this to just $1/\epsilon$ under the assumption that no two clauses are the same (Theorem 10).

Our second main result is a space lower bound on streaming algorithms that approximate the optimum of `MAX-AND-SAT` up to a factor $1/2 + \epsilon$. In particular, we show that such algorithms must use $\Omega(mn)$ space.

Theorem 2 (Main result 2)**.** *Assuming $n \geq K \lg m$ for some sufficiently large constant K, any single-pass streaming algorithm that decides if* $\texttt{OPT}(\texttt{MAX-AND-SAT}) = 1$ *or* $\texttt{OPT}(\texttt{MAX-AND-SAT}) = 2$ *with success probability at least $1 - 1/(mn)^2$ requires $\Omega(mn)$ bits of 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-2-SAT` [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-2-SAT` [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 $O(\text{poly}(1/\epsilon, \lg m))$ time using $O(n + m)$ processors [Tre98]. One of our sampling results (Lemma 5) can improve the number of processors to just $O(n)$.

Notation. We occasionally use set notation for clauses. For example, we write $x_i \in C_j$ (or $\overline{x_i} \in C_j$) if the *literal* x_i (or $\overline{x_i}$ respectively) is in clause C_j. We often write $\texttt{OPT}(P)$ to denote the optimal

value of the problem P, and when the context is clear, we drop P and just write $\texttt{OPT}$. We write $\text{poly}(x)$ to denote x^c for an arbitrarily large constant c. The constant c is absorbed in the big-O.

2 Streaming Algorithms for MAX-SAT

The framework. Without loss of generality, we may assume that there is no *trivially true* clause, i.e., clauses that contain both a literal and its negation and there are no duplicate literals in a clause. We show that if we sample $\Theta(n/\epsilon^2)$ 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 $\texttt{OPT} \geq m/2$ and a Chernoff-Union bound style argument. We use the following version of Chernoff bound.

Theorem 3 (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 any $\eta > 0$:*

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

Lemma 4. *Suppose there are more than Kn/ϵ^2 clauses in the input for some sufficiently large constant K and we run an α approximation algorithm on Kn/ϵ^2 clauses sampled uniformly at random. Then, we obtain an $\alpha - \epsilon$ approximation to* MAX-SAT *on the original input clauses with probability at least $1 - e^{-n}$.*

Proof. We recall the folklore fact that $\texttt{OPT} \geq m/2$ where m is the number of input clauses. Consider an arbitrary assignment. If it satisfies at least $m/2$ clauses, we are done. Otherwise, we can invert the assignment to satisfy at least $m/2$ clauses.

We sample Kn/ϵ^2 clauses uniformly at random for some sufficiently large constant K. Suppose that an assignment A satisfies $m_A = y$ clauses. Let the number of sampled clauses that A satisfies be m'_A and let $p = Kn/(\epsilon^2 m)$. We observe that $\mathrm{E}[m'_A] = pm_A$.

Without loss of generality, suppose the assignment A satisfies clauses $C_1, \ldots, C_y$. We define the indicator variable $X_i = [C_i \text{ is sampled}]$ and so $m'_A = \sum_{i=1}^{y} X_i$. Let $\eta = \epsilon\,\texttt{OPT}/y$. Since we sample without replacement, $\{X_i\}$ are negatively correlated. Appealing to the above Chernoff bound, we have

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

The last inequality follows because $3\,\texttt{OPT} \geq 2y + \epsilon\,\texttt{OPT}$. Therefore,

$$\begin{aligned}
\Pr\left(m'_A = pm_A \pm \epsilon p\,\texttt{OPT}\right) &\geq 1 - 2 \cdot \exp\left(-\epsilon^2 p\,\texttt{OPT}/3\right) \\
&= 1 - 2 \cdot \exp\left(-\epsilon^2 (Kn/(m\epsilon^2))\,\texttt{OPT}/3\right) \\
&\geq 1 - 2 \cdot \exp(-Kn/6) \\
&\geq 1 - \exp(-100n) .
\end{aligned}$$

The second inequality follows from the fact that $\texttt{OPT} \geq m/2$ and the last inequality holds for some sufficiently large constant K. Finally, by taking a union bound over 2^n distinct assignments, we deduce that with probability at least $1 - e^{-n}$, for all assignments A, we have $m'_A = pm_A \pm \epsilon p\,\texttt{OPT}$.

Suppose an assignment $\tilde{A}$ is an α approximation to MAX-SAT on the sampled clauses. Let A^* be an optimal assignment on the original input clauses. From the above, with probability at least $1 - e^{-n}$, we have

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

Therefore, $m_{\tilde{A}} \geq \alpha\,\texttt{OPT}(1-\epsilon) - \epsilon\,\texttt{OPT} \geq (\alpha - 2\epsilon)\,\texttt{OPT}$. We can reparameterize $\epsilon \leftarrow \epsilon/2$ to deduce the claim. □

Note that storing a clause may require $\Omega(n)$ space; thus, the space use can still be $\Omega(n^2/\epsilon^2)$ after we sample the input clauses as above. We need a second key observation. We observe 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 5. *If in an assignment, each literal is set to* `true` *independently with probability at least γ, then for some sufficiently large constant K, the assignment satisfies all $(K \lg m)/\gamma$-large clauses with probability at least $1 - 1/\operatorname{poly}(m)$.*

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

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

1 Let K be some sufficiently large constant. Ignore all β-large clauses where $\beta = (K \lg m)/\gamma$.
2 Among the remaining small clauses, sample and collect (i.e., store) $s = Kn/\epsilon^2$ clauses uniformly at random. If the number of remaining clauses is less than s, collect all of them. Let the set of collected clauses be W.
3 **Post-processing:** Run an α approximation for MAX-SAT on the collected clauses W in which each literal is set to `true` independently with probability at least γ.

Algorithm 1: A meta algorithm for sublinear MAX-SAT

Let L and S be the set of β-large and small clauses respectively. Furthermore, let $\texttt{OPT}_L$ and $\texttt{OPT}_S$ be the number of satisfied clauses in L and S respectively in the optimal assignment. Recall that W is the set of clauses stored by the algorithm.

Post-processing algorithm 1: An exponential-time $1-\epsilon$ approximation. Here, we set $\beta = (K \lg m)/\epsilon$ for some sufficiently large constant K. Suppose in post-processing, we run the exact (brute-forte) algorithm to find an optimal assignment A^* on the set of collected clauses W. Ideally, we would like to A) apply Lemma 4 to argue that we yield a $1-\epsilon$ approximation on the small clauses in S w.h.p. and B) apply Lemma 5 to argue that we satisfy all the large clauses L w.h.p.

However, the second claim requires that each literal is set to `true` with probability at least ϵ whereas 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 $x_i = q \in \{\texttt{true}, \texttt{false}\}$, we set $x_i = q$ with probability $1 - \epsilon$ (and $x_i = \overline{q}$ with probability ϵ). We will show that this yields a $1 - \epsilon$ approximation in expectation which can then be repeated $O((\log m)/\epsilon)$ times to obtain the "w.h.p." guarantee. See Algorithm 2.

```
1 Obtain an optimal assignment A* on W.
2 Let Q = ⌈(K lg m)/ε⌉ for some sufficiently large constant K.
3 for each trial t = 1, 2, ..., Q do
4 |  if x_i = q in A* then set x_i = q with probability 1 − ε in assignment A_t.
5 end for
6 Return the assignment A_t that satisfies the most number of clauses in W.
```

Algorithm 2: A post-processing algorithm that obtains a $1 - 2\epsilon$ approximation w.h.p.

Before analyzing the above post-processing algorithm, we observe that if we obtain an $\alpha \geq 1/2$ approximation in expectation, we can repeat the corresponding algorithm $O((\lg m)/\epsilon)$ times and choosing the best solution to obtain an $\alpha - \epsilon$ approximation w.h.p.

Lemma 6. *Suppose there is an algorithm that yields an $\alpha \geq 1/2$ approximation to* `MAX-SAT` *in expectation. By repeating the algorithm $O((\lg m)/\epsilon)$ times and choosing the best solution, we yield an $\alpha - \epsilon$ approximation with probability at least $1 - 1/\operatorname{poly}(m)$.*

Proof. Let Z be the number of clauses satisfied by the algorithm. First, note that since $m/4 \leq \alpha\,\texttt{OPT}$ (since $\alpha \geq 1/2$), we have $\alpha\,\texttt{OPT}\,/(m - \alpha\,\texttt{OPT}) \geq 1/3$. We have $\mathrm{E}\,[m - Z] \leq m - \alpha\,\texttt{OPT}$. Appealing to Markov inequality,

$$\begin{aligned}\Pr\left(Z \leq (1-\epsilon)\alpha\,\texttt{OPT}\right) &= \Pr\left(m - Z \geq m - (1-\epsilon)\alpha\,\texttt{OPT}\right) \\ &\leq \frac{m - \alpha\,\texttt{OPT}}{m - (1-\epsilon)\alpha\,\texttt{OPT}} \\ &= \frac{1}{1 + \epsilon\alpha\,\texttt{OPT}\,/(m - \alpha\,\texttt{OPT})} \leq \frac{1}{1+\epsilon/3}\,.\end{aligned}$$

Hence, we can repeat the algorithm $O(\lg_{1+\epsilon/3} m) = O(1/\epsilon \cdot \lg m)$ times and choose the best solution to obtain an $\alpha - \epsilon$ approximation with probability at least $1 - 1/\operatorname{poly}(m)$. □

Now, we are ready to analyze the post-processing Algorithm 2.

Lemma 7. *We have the following.*

1. *The assignment returned by post-processing Algorithm 2 satisfies all clauses in L with probability at least $1 - 1/\operatorname{poly}(m)$;*
2. *Algorithm 2 yields a $1 - 2\epsilon$ approximation on S w.h.p.*

These two claims imply that Algorithm 2 yields a $1 - 2\epsilon$ approximation on the original input w.h.p.

Proof. Clearly, in each trial $t = 1, 2, \ldots, Q$, each literal is set to `true` with probability at least ϵ. According to Lemma 5, each assignment A_t satisfies all the large clauses in L with probability at least $1 - 1/\operatorname{poly}(m)$. Taking a union bound over $Q = O((K \lg m)/\epsilon)$ trials, we conclude that all assignments A_t satisfy all the clauses in L with probability at least $1 - 1/\operatorname{poly}(m)$.

We now prove the second claim. Let B be the set of clauses in W satisfied by the exact assignment A^*. Consider any trial t. By linearity of expectation, the expected number of satisfied clauses in B after we randomly perturb the exact assignment is

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

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 $1-\epsilon$. Appealing to Lemma 6, the assignment returned by post-processing Algorithm 2 yields a $1-2\epsilon$ approximation on W with probability at least $1 - 1/\operatorname{poly}(m)$. This, in turn, implies that we obtain a $1-3\epsilon$ approximation on S with probability at least $1 - 1/\operatorname{poly}(m) - e^{-n} \geq 1 - 1/\operatorname{poly}(m) - 1/\operatorname{poly}(n)$ by Lemma 4.

Therefore, we satisfy at least $(1-3\epsilon)\,\texttt{OPT}_S + \texttt{OPT}_L \geq (1-3\epsilon)\,\texttt{OPT}$ clauses with probability at least $1 - 1/\operatorname{poly}(m) - 1/\operatorname{poly}(n)$. □

Post-processing algorithm 2: A polynomial-time $3/4-\epsilon$ approximation. We can set $\beta = K \lg m$ for some sufficiently large constant K 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] for a `MAX-SAT` instance with m clauses and n variables.

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

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

Lemma 8 ([GW94], Theorem 5.3)**.** *The optimal fractional solution of the linear program for* `MAX-SAT` *can be rounded to yield an assignment that satisfies* $3/4 \cdot \texttt{OPT}(LP) \geq 3/4 \cdot \texttt{OPT}$ *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.*

```
1 Obtain an optimal solution z*, y* for the linear program of MAX-SAT on W.
2 Let Q = ⌈(K lg m)/ε⌉ for some sufficiently large constant K.
3 for each trial t = 1, 2, ..., Q do
4 |   Set x_i = true with probability 1/4 + y_i*/2 in assignment A_t.
5 end for
6 Return the assignment A_t that satisfies the most number of clauses in W.
```

Algorithm 3: A post-processing algorithm that obtains a $3/4-2\epsilon$ approximation w.h.p.

The following Lemma is more or less similar to Lemma 7.

Lemma 9. *We have the following.*

1. *The assignment returned by post-processing Algorithm 3 satisfies all clauses in L with probability at least $1 - 1/\operatorname{poly}(m)$;*

2. *Algorithm 2 yields a $3/4 - 2\epsilon$ approximation on S w.h.p.*

These two claims imply that Algorithm 3 yields a $3/4 - 2\epsilon$ approximation on the original input w.h.p.

Proof. In each trial $t = 1, 2, \ldots, Q$, each literal is set to `true` with probability at least $1/4$. This is because $1/4 \leq 1/4 + y_i^*/2 \leq 3/4$. Appealing to Lemma 5 with $\gamma = 1/4$, each assignment A_t satisfies all the large clauses in L with probability at least $1 - 1/\operatorname{poly}(m)$. Taking a union bound over Q trials, we conclude that all assignments A_t satisfy all the clauses in L with probability at least $1 - 1/\operatorname{poly}(m)$.

The proof of the second claim is analogous to that of Lemma 7. □

Putting it all together. 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. We ignore β-large clauses during the stream. Among the remaining clauses, we can sample and store $s = \Theta(n/\epsilon^2)$ small clauses in the stream uniformly at random using Reservoir sampling [Vit85]. Since we collect at most $T = \min\{m, n/\epsilon^2\}$ clauses, each of which has size at most β, the space use is therefore $O(T\beta)$.

We can run post-processing Algorithm 2 where $\beta = (K \lg m)/\epsilon$ and yield a $1 - \epsilon$ approximation. Alternatively, we set $\beta = (K \lg m)$ and run the post-processing Algorithm 3 to yield a $3/4 - \epsilon$ approximation. □

We remark that it is possible to have an approximation slightly 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. The analysis is analogous to what we outlined above. If we are satisfied with an approximation only in expectation, we can also drop the $\lg m$ factor in β.

An improvement under the no-duplicate assumption. In practice, we normally run the $3/4 - \epsilon$ approximation algorithm described above. However, the memory use does not scale well for small ϵ. We present an improvement that saves another $1/\epsilon$ factor under the assumption that no two clauses are the same. In fact, it suffices to make this assumption for 1-literal clauses.

Theorem 10. *Suppose there are no duplicate clauses. There exists a polynomial-time, $O(n/\epsilon \cdot \lg m)$-space algorithm, single-pass streaming algorithm that yields a $3/4 - \epsilon$ approximation to* `MAX-SAT` *w.h.p.*

Proof. If the number of 1-literal clauses is at most ϵm, we can safely ignore these clauses. This is because the number of 1-literal clauses is then at most $2\epsilon\,\texttt{OPT}$. If we randomly set each variable to `true` with probability $1/2$, we will yield a $3/4$ approximation in expectation on the remaining clauses since each of these clause is satisfied with probability at least $1 - 1/2^2 = 3/4$. Appealing to Lemma 6, we can run $O((\lg m)/\epsilon)$ copies in parallel and return the best assignment to yield a $3/4 - \epsilon$ approximation w.h.p. Therefore, the overall approximation is $3/4 - 3\epsilon$ and the space use is $O(n/\epsilon \cdot \lg m)$.

If the number of 1-literal clauses is more than ϵm, then we know that $m \leq 2n/\epsilon$ since the number of 1-literal clauses is at most $2n$ if there are no duplicate clauses. Therefore, we can run the first algorithm in Theorem 1 which uses $O(n/\epsilon \cdot \lg m)$ space.

While we do not know which case the input is in advance, we can simply run the two algorithms above in parallel and simply terminate the second algorithm if $m > 2n/\epsilon$. □

3 Lower Bounds

Lower bound for MAX-SAT. The streaming maximum cut problem asks to output an approximation of the size of the maximum cut $\texttt{OPT}(\texttt{MAX-CUT}) = \max_{S \subset V} |E(S, V \setminus S)|$ of a graph $G = (V, E)$. In this problem, the stream consists of edges of the underlying graph. We will show that one can use MAX-SAT to encode the maximum cut problem.

Kapralov and Krachun [KK19] showed that for any $\epsilon \in (1/n^{1/10}, 1)$ deciding if $\texttt{OPT}(\texttt{MAX-CUT}) \geq m_0$ or $\texttt{OPT}(\texttt{MAX-CUT}) \leq m_0/(2-\epsilon)$ with success probability at least 99/100 requires $\Omega(|V|)$ space for some m_0.

We also note that $|E| \leq 2\,\texttt{OPT}(\texttt{MAX-CUT})$ (this follows from a probabilistic argument that randomly partitions the vertices into two equal-size parts). Thus, $|E|/2 \leq m_0 \leq |E|$. Let $\delta = m_0/|E|$; note that $1 - \epsilon/2 \leq \delta \leq 1$. Note that $\delta \geq 1 - \epsilon/2$ because in the second case we must have $|E|/2 \leq \texttt{OPT}(\texttt{MAX-CUT}) \leq m_0/(2-\epsilon)$ which implies $m_0/|E| \geq 1 - \epsilon/2$.

Theorem 11. *For any $\epsilon \in (n^{-1/10}, 1)$, there is some fixed δ, where $1 - \epsilon/2 \leq \delta \leq 1$, such that any single-pass streaming algorithm that decides if $\texttt{OPT}(\texttt{MAX-SAT}) \leq m/2 \cdot (1 + \delta/(2-\epsilon))$ or $\texttt{OPT}(\texttt{MAX-SAT}) \geq m/2 \cdot (1+\delta)$ with success probability at least 0.99 requires $\Omega(n)$ bits of space.*

Proof. Consider the cut between S and $V \setminus S$. We set $x_u = \texttt{true}$ if and only if $u \in S$. When an edge uv arrives in the graph stream, we put two clauses $(x_u \vee x_v)$ and $(\overline{x_u} \vee \overline{x_v})$ in the stream. Note that there are $n = |V|$ variables corresponding to vertices in the graph and there are $m = 2|E|$ clauses.

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. It is easy to see that

$$\texttt{OPT}(\texttt{MAX-SAT}) = |E| + \texttt{OPT}(\texttt{MAX-CUT})\ .$$

We have

$$\texttt{OPT}(\texttt{MAX-CUT}) \geq m_0 \implies \texttt{OPT}(\texttt{MAX-SAT}) \geq |E| + m_0 = |E|(1+\delta);$$
$$\texttt{OPT}(\texttt{MAX-CUT}) \leq \frac{m_0}{2-\epsilon} \implies \texttt{OPT}(\texttt{MAX-SAT}) \leq |E| + \frac{m_0}{2-\epsilon} = |E|(1 + \delta/(2-\epsilon))\ .$$

Thus, if we can decide the two aforementioned cases, we can solve the decision problem of MAX-CUT which requires $\Omega(n)$ space. Hence, the lower bound follows. □

We note that this implies that an approximation better than $\frac{1+1/(2-\epsilon)}{2-\epsilon} \approx 3/4 + O(\epsilon)$ requires $\Omega(n)$ space.

Lower bound for MAX-AND-SAT. Our second main result is to prove Theorem 2. That is, we show that MAX-AND-SAT does not admit a sublinear space streaming algorithm that decides if $\texttt{OPT} = 1$ or $\texttt{OPT} = 2$. In our lower bound, $n = \Omega(\lg m)$.

Consider the following one-way communication problem. Alice receives a length-nm random bitstring A (each bit is either 0 or 1 equiprobably) that Bob wants to learn about say with success probability at least $1 - 1/\operatorname{poly}(nm)$. Alice must send a message of length $\Omega(mn)$ bits 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 $i \in [n]$. Newman [New91] showed that we can translate a protocol using public randomness with success probability at least $1 - 1/\operatorname{poly}(mn)$ to a protocol using private randomness with success probability at least $1 - 1/\operatorname{poly}(mn)$ using only $O(\lg(nm))$ extra bits of communication.

Alice constructs her part of the stream as follows. 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 $Z_{ji} = 0$. If $A_{ji} = 0$, then C_j contains neither x_i nor $\overline{x_i}$. More formally,

$$C_j = \Big(\bigwedge_{i:A_{ji}=1, Z_{ji}=1} x_i \Big) \Big(\bigwedge_{i:A_{ji}=1, Z_{ji}=0} \overline{x_i} \Big) .$$

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 12. *Suppose $n \geq K \lg m$ for some sufficiently large constant K. With probability at least $1 - 1/\operatorname{poly}(nm)$, exactly 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_j 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 $\Pr(x_i \in C_j \wedge \overline{x_i} \in C_k) = 1/16$. This is because $\Pr(A_{ji} = A_{ki} = Z_{ji} = 1 \wedge Z_{ki} = 0) = 1/2^4$.

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 exactly one clause among $C_1, \ldots, C_m$ can be satisfied is at least

$$1 - \binom{m}{2}(15/16)^n = 1 - \binom{m}{2}(15/16)^{n/2+n/2} \geq 1 - \frac{1}{1.01^n \cdot \operatorname{poly}(m)} \geq 1 - 1/\operatorname{poly}(nm)$$

where we used the assumption that $n \geq K \lg m$ for some sufficiently large constant K and $(16/15)^{1/2} > 1.01$ to yield the first inequality. □

Alice then sends the memory of the streaming algorithm to Bob. Recall that Bob wants to recover the bitstring A. Let $C^{(ji)}$ be a clause in which: a) for all $i' \neq i$, $x_{i'} \in C^{(ji)}$ if $Z_{ji'} = 1$ and $\overline{x_{i'}} \in C^{(ji)}$ if $Z_{ji'} = 0$ and b) $x_i \in C^{(ji)}$ if $Z_{ji} = 0$ and $\overline{x_i} \in C^{(ji)}$ if $Z_{ji} = 1$. More formally,

$$C^{(ji)} = \begin{cases} \Big(\bigwedge_{i' \neq i : Z_{ji'}=1} x_{i'} \Big) \Big(\bigwedge_{i' \neq i : Z_{ji'}=0} \overline{x_{i'}} \Big) \bigwedge x_i & \text{if } Z_{ji} = 0 \\ \Big(\bigwedge_{i' \neq i : Z_{ji'}=1} x_{i'} \Big) \Big(\bigwedge_{i' \neq i : Z_{ji'}=0} \overline{x_{i'}} \Big) \bigwedge \overline{x_i} & \text{if } Z_{ji} = 1 . \end{cases}$$

The following lemma shows that $C^{(ji)}$ cannot be satisfied simultaneously with another C_l where $l \neq j$. Its proof is almost identical to that of Lemma 12.

Lemma 13. *Suppose $n \geq K \lg m$ for some sufficiently large constant K. For fixed $j \in [m]$ and $i \in [n]$, for all $l \neq j$, the clauses C_l and $C^{(ji)}$ cannot be both satisfied with probability at least $1 - 1/\operatorname{poly}(mn)$.*

Proof. The sets of literals in $C^{(ji)}$ and C_l are independent of each other. Each literal x_k (or $\overline{x_k}$) appears in $C^{(ji)}$ with probability $1/2$ and appears in C_l with probability $1/4$ and the two events

are independent since Z_{jk}, A_{lk}, Z_{lk} are independent. Thus, the probability that there does not exist $k \in [n]$ such that $x_k \in C^{(ji)}$ 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 $n \geq K \lg m$, we deduce that no other clause C_l (where $l \neq j$) can be satisfied simultaneously with $C^{(ji)}$ with probability at least $1 - 1/(1.01^n \cdot \text{poly}(m)) \geq 1 - 1/\text{poly}(mn)$. □

The following lemma allows Bob to correctly recover the bitstring.

Lemma 14. *Consider the stream $C_1, \ldots, C_m, C^{(ji)}$. With probability at least $1 - 1/\text{poly}(nm)$, if $A_{ji} = 0$, then $\texttt{OPT} = 2$ and if $A_{ji} = 1$, then $\texttt{OPT} = 1$.*

Proof. Note that for every variable $x_{i'}$ where $i' \neq i$, if it appears in both C_j and $C^{(ji)}$, it appears the same way (negated or unnegated).

If $A_{ji} = 0$, then neither the literal x_i nor the literal $\overline{x_i}$ is in C_j and therefore there does not exist any variable that appears differently in C_j and $C^{(ji)}$. This implies that C_j and $C^{(ji)}$ can be both satisfied.

If $A_{ji} = 1$, then we know that the variable x_i appears in C_j. But only one clause among C_j and $C^{(ji)}$ can now be satisfied since x_i appears negated in one clause and unnegated in the other due to our construction.

Combining with Lemma 12 and Lemma 13, the claim follows. □

Based on Lemma 14, the procedure in Algorithm 4 allows Bob to recover A.

```
for j = 1, 2, ..., m do
    for i = 1, 2, ..., n do
        Use algorithm A to check if OPT = 1 on the stream C_1, ..., C_m, C^(ji).
        if OPT = 1 then A_ji = 1
        else A_ji = 0
    end for
end for
```

Algorithm 4: Procedure to recover A

Hence, Bob can recover A_{ji} for each $i \in [n]$ and $j \in [m]$ by using algorithm $\mathcal{A}$ on the stream $C_1, \ldots, C_m, C^{(ji)}$. This is possible since Bob has the memory state of the algorithm for the stream $C_1, \ldots, C_m$ from Alice. Since the algorithm failure probability is at most $1/(nm)^2$, the overall failure probability of the recover procedure is at most

$$nm\left(1/(nm)^2 + 1/\text{poly}(nm)\right) \leq 2/(nm) \ .$$

If the streaming algorithm $\mathcal{A}$ uses $o(nm)$ bits of memory, Bob can distinguish 2^{nm} different inputs using $o(nm)$ bits of communication w.h.p. which is a contradiction. This completes the proof of Theorem 2.

References

[ABL+10] Josep Argelich, Daniel Le Berre, Inês Lynce, João P. Marques Silva, and Pascal Rapicault. Solving linux upgradeability problems using boolean optimization. In *LoCoCo*, volume 29 of *EPTCS*, pages 11–22, 2010.

[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.

[AM07] Josep Argelich and Felip Manyà. Partial max-sat solvers with clause learning. In *SAT*, volume 4501 of *Lecture Notes in Computer Science*, pages 28–40. Springer, 2007.

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

[CSMV10] Yibin Chen, Sean Safarpour, João Marques-Silva, and Andreas G. Veneris. Automated design debugging with maximum satisfiability. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 29(11):1804–1817, 2010.

[Cus08] James Cussens. Bayesian network learning by compiling to weighted MAX-SAT. In *UAI*, pages 105–112. AUAI Press, 2008.

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[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.

[HPJ+17] Antti Hyttinen, Sergey M. Plis, Matti Järvisalo, Frederick Eberhardt, and David Danks. A constraint optimization approach to causal discovery from subsampled time series data. *Int. J. Approx. Reason.*, 90:208–225, 2017.

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

[JSS00] Daniel Jackson, Ian Schechter, and Ilya Shlyakhter. Alcoa: the alloy constraint analyzer. In *ICSE*, pages 730–733. ACM, 2000.

[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.

[LM06] Inês Lynce and João Marques-Silva. Efficient haplotype inference with boolean satisfiability. In *AAAI*, pages 104–109. AAAI Press, 2006.

[MAX] Maxsat evaluation 2019. `https://maxsat-evaluations.github.io/2019/benchmarks.html`.

[MJIM15] João Marques-Silva, Mikolás Janota, Alexey Ignatiev, and António Morgado. Efficient model based diagnosis with maximum satisfiability. In *IJCAI*, pages 1966–1972. AAAI Press, 2015.

[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.

[SZGN17] Xujie Si, Xin Zhang, Radu Grigore, and Mayur Naik. Maximum satisfiability in software analysis: Applications and techniques. In *CAV (1)*, volume 10426 of *Lecture Notes in Computer Science*, pages 68–94. Springer, 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.