Massively Parallel Maximum Coverage Revisited*

Thai Bui and Hoa T. Vu

San Diego State University, San Diego, CA 92182, USA
`tbui8182@sdsu.edu, hvu2@sdsu.edu`

Abstract. We study the maximum set coverage problem in the massively parallel model. In this setting, m sets that are subsets of a universe of n elements are distributed among m machines. In each round, these machines can communicate with each other, subject to the memory constraint that no machine may use more than $\tilde{O}(n)$ memory. The objective is to find the k sets whose coverage is maximized. We consider the regime where $k = \Omega(m)$ (i.e., $k = m/100$), $m = O(n)$, and each machine has $\tilde{O}(n)$ memory [1].

Maximum coverage is a special case of the submodular maximization problem subject to a cardinality constraint. This problem can be approximated to within a $1 - 1/e$ factor using the greedy algorithm, but this approach is not directly applicable to parallel and distributed models. When $k = \Omega(m)$, to obtain a $1 - 1/e - \epsilon$ approximation, previous work either requires $\tilde{O}(mn)$ memory per machine which is not interesting compared to the trivial algorithm that sends the entire input to a single machine, or requires $2^{O(1/\epsilon)}n$ memory per machine which is prohibitively expensive even for a moderately small value ϵ.

Our result is a randomized $(1 - 1/e - \epsilon)$-approximation algorithm that uses

$$O(1/\epsilon^3 \cdot \log m \cdot (\log(1/\epsilon) + \log m))$$

rounds. Our algorithm involves solving a slightly transformed linear program of the maximum coverage problem using the multiplicative weights update method, classic techniques in parallel computing such as parallel prefix, and various combinatorial arguments.

1 Introduction

Maximum coverage is a classic NP-Hard problem. In this problem, we have m sets $S_1, S_2, \ldots, S_m$ that are subsets of a universe of n elements $[n] = \{1, 2, \ldots, n\}$. The goal is to find k sets that cover the maximum number of elements. In the offline model, the greedy algorithm achieves a $1 - 1/e$ approximation and assuming $\mathrm{P} \neq \mathrm{NP}$, this approximation is the best possible in polynomial time [8].

However, the greedy algorithm for maximum coverage and the related set cover problem is not friendly to streaming, distributed, and massively parallel

* This work is supported by the National Science Foundation under Grant No. 2342527.

[1] The input size is $O(mn)$ and each machine has the memory enough to store a constant number of sets.

computing. A large body of work has been devoted to designing algorithms for these problems in these big data computation models. An incomplete list of work includes [3–7, 9, 11–13, 16, 21, 22, 24, 25].

Some example applications of maximum coverage includes facility and sensor placement [17], circuit layout and job scheduling [10], information retrieval [1], market design [15], data summarization [24], and social network analysis [13].

The MPC model. We consider the massively parallel computation model (MPC) in which m sets $S_1, S_2, \ldots, S_m \subseteq [n]$ are distributed among m machines. Each machine has memory $\tilde{O}(n)$ and holds a set. In each round, each machine can communicate with others with the constraint that no machine receives a total message of size more than $\tilde{O}(n)$. Similar to previous work in the literature, we assume that $m \leq n$.

The MPC model, introduced by Karloff, Suri, and Vassilvitskii [14] is an abstraction of various modern computing paradigms such as MapReduce, Hadoop, and Spark.

Previous work. This problem is a special case of submodular maximization subject to a cardinality constraint. The results of Liu and Vondrak [20], Barbosa et al. [23], Kumar et al. [18] typically require that each machine has enough memory to store $O(\sqrt{km})$ items which are sets in our case (and storing a set requires $\tilde{O}(n)$ memory) with $\sqrt{m/k}$ machines. When $k = \Omega(m)$ (e.g., $k = m/100$), this means that a single machine may need $\tilde{O}(mn)$ memory. This is not better than the trivial algorithm that sends the entire input to a single machine and solves the problem in 1 round.

Assadi and Khanna gave a randomized $1 - 1/e - \epsilon$ approximation algorithm in which each machine has $\tilde{O}\left(m^{\delta/\epsilon} n\right)$ memory and the number of machines is $m^{1-\delta/\epsilon}$ for any $\epsilon, \delta \in (0, 1)$ (see Corollary 10 in the full paper of [4]). Setting $\delta = \Theta(1/\log m)$ gives us a $1 - 1/e - \epsilon$ approximation in $O(1/\epsilon \cdot \log m)$ rounds with $O(m)$ machines each of which uses $\tilde{O}\left(2^{1/\epsilon} n\right)$ memory. While Assadi and Khanna's result is nontrivial in this regime, the dependence on ϵ is exponential and if n is large, then even a moderately small value of $\epsilon = 0.01$ can lead to a prohibitively large memory requirement $\approx 2^{100} n$. Their work however can handle the case where $k = o(m)$.

Our result. We present a relatively simple randomized algorithm that achieves a $1 - 1/e - \epsilon$ approximation in $O(1/\epsilon^3 \cdot \log m \cdot (\log(1/\epsilon) + \log m))$ rounds with $\tilde{O}(n)$ memory per machine assuming $k = \Omega(m)$. Our space requirement does not depend on ϵ compared to the exponential dependence in Assadi and Khanna's result.

We note that assuming $k = \Omega(m)$ does not make the problem any easier since there are still exponentially many solutions to consider. In practice, one can think of many applications where one can utilize a constant fraction of the available sets (e.g., 10% or 20%). We state our main result as a theorem below.

Theorem 1. *Assume $k = \Omega(m)$ and there are m machines each of which has $\tilde{O}(n)$ memory. There exists an algorithm that with high probability finds k sets that cover at least $(1-1/e-\epsilon)\mathrm{OPT}$ elements in $O(1/\epsilon^3 \cdot \log m \cdot (\log(1/\epsilon)+\log m))$ rounds.*

If the maximum frequency f (the maximum number of sets that any element belongs to) is bounded, we can drop the assumption that $k = \Omega(m)$, and parameterize the number of rounds based on f. In particular, we can obtain a $1 - 1/e - \epsilon$ approximation in $O(f^3/\epsilon^6 \cdot \log^2(\frac{kf}{\epsilon}))$ rounds.

Remark. We could easily modify our algorithm so that each machine uses $\tilde{O}(1/\epsilon \cdot n)$ memory and the number of rounds is $O(1/\epsilon^2 \cdot \log m \cdot (\log(1/\epsilon)+\log m))$. At least one $\log m$ factor is necessary based on the lower bound given by Corollary 9 of [4].

Randomization appears in two parts of our algorithms: the rounding step and the subsampling step to reduce the number of rounds from $\log m \cdot \log n$ to $\log m \cdot (\log(1/\epsilon) + \log m)$. If we only need to compute an approximation to the optimal coverage value such that the output is in the interval $[(1-\epsilon)\mathrm{OPT}, \mathrm{OPT}/(1-1/e-\epsilon)]$, then we have a deterministic algorithm that runs in $O(1/\epsilon^3 \cdot \log n \cdot \log m)$ rounds. The algorithm by Assadi and Khanna [4] combines the sample-and-prune framework with threshold greedy. This strategy requires sampling sets. It is unclear how to derandomize their algorithm even just to compute an approximation to the optimal coverage value.

Our techniques and paper organization. In Section 2.1, we transform the standard linear program for the maximum coverage problem into an equivalent packing linear program that can be solved "approximately" by the multiplicative weights update method. At a high level, the multiplicative weights update method gives us a fractional solution that is a $1-1/e-O(\epsilon)$ bi-criteria approximation where $(1+O(\epsilon))k$ "fractional" sets cover $(1-1/e-O(\epsilon))\mathrm{OPT}$ "fractional" elements. We then show how to find k sets covering $(1-1/e-O(\epsilon))\mathrm{OPT}$ elements from this fractional solution through a combinatorial argument and parallel prefix.

Section 2.2 outlines the details to solve the transformed linear program in the MPC model. While this part is an adaptation of the standard multiplicative weights, an implementation in the MPC model requires some additional details such as the number of bits to represent the weights. All missing proofs and detailed calculation can be found in the full version.

Preliminaries. In this work, we will always consider the case where each machine has $\tilde{O}(n)$ memory and $m \leq n$. Without loss of generality, we may assume the non-central machine j stores the set S_j. For each element $i \in [n]$, we use f_i to denote the number of sets that i is in. This is also referred to as the frequency of i. Assume each machine has $\tilde{O}(n)$ space. The vector $\mathbf{f}$ can be computed in $O(\log m)$ rounds and broadcasted to all machines. Each machine j starts with the characteristic vector $v_j \in \{0,1\}^n$ of the set S_j that it holds. The vector $\mathbf{f}$ is just the sum of the characteristic vectors of the sets. We can aggregate the vectors $\{v_j\}$ in $O(\log m)$ rounds using the standard binary tree aggregation algorithm.

Since in this work, the dependence on $1/\epsilon$ is polynomial, an $\alpha - O(\epsilon)$ approximation can easily be translated to an $\alpha - \epsilon$ approximation by scaling ϵ by a constant factor. We can also assume that $1/\epsilon < n/10$; otherwise, we can simulate the greedy algorithm in $O(1/\epsilon)$ rounds. For the sake of exposition, we will not attempt to optimize the constants in our algorithm and analysis. Finally, in this work we consider $1 - 1/\text{poly}(m)$ as a "high probability". We use $[E]$ to denote the indicator variable of the event E that is 1 if E happens and 0 otherwise.

2 Algorithm

2.1 The main algorithm

Linear programming (re)formulation. We first recall the relaxed linear program (LP) for the maximum coverage problem Π_0:

$$\begin{aligned}
\text{maximize} \quad & \sum_{i\in[n]} x_i \\
\text{(s.t.)} \quad & x_i \le \sum_{S_j \ni i} y_j && \forall i \in [n] \\
& \sum_{j\in[m]} y_j = k \\
& x_i, y_j \in [0,1] && \forall i \in [n], j \in [m].
\end{aligned}$$

We first reformulate this LP and then approximately solve the new LP using the multiplicative weights update method [2]. For each $j \in [m]$, let $z_j := 1 - y_j$. We have the following fact.

Fact 1 *For each $i \in [n]$, $x_i \le \sum_{S_j \ni i} y_j \iff x_i + \sum_{S_j \ni i} z_j \le \sum_{S_j \ni i} (y_j + z_j) = f_i$.*

Note that if $\mathbf{y} \in [0,1]^m$ and $\sum_j y_j = k$, then $\mathbf{z} \in [0,1]^m$ and $\sum_j z_j = m - k$. Thus, it is not hard to see that the original LP is equivalent to the following LP which we will refer to as Π_1.

$$\begin{aligned}
\text{maximize} \quad & \sum_{i\in[n]} x_i \\
\text{(s.t.)} \quad & \frac{x_i}{f_i} + \frac{1}{f_i} \cdot \sum_{S_j \ni i} z_j \le 1 && \forall i \in [n] \\
& \sum_{j\in[m]} z_j = m - k \\
& x_i, z_j \in [0,1] && \forall i \in [n], j \in [m].
\end{aligned}$$

In this section, we will assume the existence an MPC algorithm that approximately solves the linear program Π_1 in $O(1/\epsilon^3 \cdot \log n \cdot \log m)$ rounds. The proof will be deferred to Section 2.2.

Theorem 2. *There is an algorithm that finds* $\boldsymbol{x} \in [0,1]^n, \boldsymbol{z} \in [0,1]^m$ *such that*

1. $\sum_{i \in [n]} x_i \geq (1-\epsilon)\mathrm{OPT}$,
2. $\sum_{j \in [m]} z_j = m - k$, *and*
3. $\frac{x_i}{f_i} + \frac{1}{f_i} \cdot \sum_{S_j \ni i} z_j \leq 1 + \epsilon \quad \forall i \in [n]$.

in $O(\epsilon^{-3} \log n \cdot \log m)$ *rounds.*

Let $\mathbf{x}$ and $\mathbf{z}$ be the be the output given by Theorem 2. Then, let $\mathbf{x}' = \mathbf{x}/(1+\epsilon)$, $\mathbf{z}' = \mathbf{z}/(1+\epsilon)$, and $\mathbf{y}' = \mathbf{1} - \mathbf{z}'$. We have

$$\sum_{i=1}^{n} x_i' = \frac{1}{1+\epsilon}\sum_{i=1}^{n} x_i \geq \frac{1-\epsilon}{1+\epsilon}\mathrm{OPT} > (1-4\epsilon)\mathrm{OPT}, \tag{1}$$

$$\sum_{j=1}^{m} y_i' = \sum_{j=1}^{m}\left(1 - \frac{z_j}{1+\epsilon}\right) = m - \frac{m-k}{1+\epsilon} \leq m - (1-2\epsilon)(m-k) < k + 2\epsilon m, \tag{2}$$

$$x_i' + \sum_{S_j \ni i} z_j' \leq f_i \iff x_i' \leq \sum_{S_j \ni i} y_j', \quad \forall i \in [n], \text{ by Fact 1.} \tag{3}$$

Thus, by setting $\mathbf{x} \leftarrow \mathbf{x}/(1+\epsilon)$, and $\mathbf{y} \leftarrow \mathbf{y}'$, we have an approximate solution $\mathbf{x} \in [0,1]^n, \mathbf{y} \in [0,1]^n$ to the LP Π_0 such that

$$\sum_{i=1}^{n} x_i \geq (1-4\epsilon)\mathrm{OPT}, \qquad \sum_{j=1}^{m} y_j \leq k + 2\epsilon m, \text{ and}$$
$$x_i \leq \sum_{S_j \ni i} y_j, \forall i \in [n].$$

We can then apply the standard randomized rounding to find a sub-collection of at most $k + 2\epsilon m$ sets that covers at least $(1-4\epsilon)\mathrm{OPT}$ elements. For the sake of completeness, we will provide the rounding algorithm in the MPC model in the following lemma.

Lemma 1. *Suppose* $\boldsymbol{x} \in [0,1]^n$ *and* $\boldsymbol{y} \in [0,1]^m$ *satisfy:*

1. $\sum_{i \in [n]} x_i \geq L$,
2. $x_i \leq \sum_{S_j \ni i} y_j$ *for all* $i \in [n]$,
3. $\sum_{j \in [m]} y_j = k$,
4. $x_i, y_j \in [0,1]$ *for all* $i \in [n]$ *and* $j \in [m]$.

Then there exists a rounding algorithm that finds a sub-collection of k *sets that in expectation cover at least* $(1-1/e)L$ *elements in* $O(1)$ *round. To obtain a high probability guarantee, the algorithm requires* $O(1/\epsilon \cdot \log m)$ *rounds to find* k *sets that cover least* $(1-1/e-O(\epsilon))L$ *elements.*

Applying Lemma 1 to **x** and **y** with $k + 2\epsilon m$ in place of k, we obtain a sub-collection of at most $k + 2\epsilon m$ sets that covers at least $(1 - 1/e - O(\epsilon))$OPT elements. Since we assume that $k = \Omega(m)$, that means we have found $k + O(\epsilon)k$ sets that cover at least $(1 - 1/e - O(\epsilon))$OPT elements. The next lemma shows that we can find k sets among these that cover at least $(1 - 1/e - O(\epsilon))$OPT elements. The proof is a combination of a counting argument and the well-known parallel prefix algorithm [19].

Algorithm 1: Parallel prefix coverage

```
1  Compute $|S_1|, |S_2 \setminus S_1|, |S_3 \setminus (S_1 \cup S_2)|, \ldots, |S_k \setminus (S_1 \cup \ldots \cup S_{k-1})|$ in $O(\log k)$
     rounds.
2  Function PrefixCoverage($S_1, S_2, \ldots, S_k$):
       // Compute $|S_1|, |S_2 \cup S_1|, |S_3 \cup S_2 \cup S_1|, \ldots, |S_k \cup S_{k-1} \cup \ldots \cup S_1|$
3      if $k = 1$ then
4          return $|S_1|$.
5      else
           // Assume $k$ is even.
6          In one round, machine $2j - 1$ sends $S_{2j-1}$ to machine $2j$, then machine
             $2j$ computes $Q_j = S_{2j-1} \cup S_{2j}$.
7          Run PrefixCoverage($Q_1, Q_2, \ldots, Q_{k/2}$) on machines $2, 4, 6, \ldots, k$.
8          Machine $j$ now has $S_1 \cup S_2 \cup S_3 \cup \ldots \cup S_j$ for $j = 2, 4, 6, \ldots, k$.
9          In one round, machine $j = 1, 3, 5, \ldots, k - 1$ communicates with
             machine $j - 1$ which has $S_1 \cup S_2 \cup \ldots S_{j-1}$ and computes
             $S_1 \cup S_2 \cup \ldots \cup S_j$.
           // If $k$ is odd, run the above algorithm on $S_1, S_2, \ldots, S_{k-1}$ and
              then compute $S_1 \cup S_2 \cup \ldots \cup S_k$ in one round.
10         Each machine $j$ communicates with machine $j - 1$ to compute
             $|S_1 \cup S_2 \cup \ldots \cup S_j| - |S_1 \cup S_2 \cup \ldots \cup S_{j-1}|$ in one round.
```

We rely on the following result which is a simulation of the parallel prefix.

Lemma 2. *Suppose there are k sets and machine j holds the set S_j. Then Algorithm 1 computes $|S_1|, |S_2 \setminus S_1|, |S_3 \setminus (S_1 \cup S_2)|, |S_4 \setminus (S_1 \cup S_2 \cup S_3)|, \ldots, |S_k \cup S_{k-1} \cup \ldots \cup S_1|$ in $O(\log k)$ rounds.*

Proof. We first show how to compute $(S_1), (S_1 \cup S_2), (S_1 \cup S_2 \cup S_3), \ldots, (S_1 \cup S_2 \cup \ldots \cup S_k)$ in $O(\log k)$ rounds where machine j holds $S_1 \cup S_2 \cup \ldots \cup S_j$ at the end. Once this is done, machine j can send $S_1 \cup S_2 \cup \ldots \cup S_j$ to machine $j + 1$ and machine $j + 1$ can compute $|S_1 \cup S_2 \cup \ldots \cup S_{j+1}| - |S_1 \cup S_2 \cup \ldots \cup S_j| = |S_{j+1} \setminus (S_1 \cup S_2 \cup \ldots \cup S_j)|$.

The algorithm operates recursively. In one round, machine $2j-1$ sends S_{2j-1} to machine $2j$, then machine $2j$ computes $Q_j = S_{2j-1} \cup S_{2j}$. Assuming k is even, the algorithm recursively computes $(Q_1), (Q_1 \cup Q_2), (Q_1 \cup Q_2 \cup Q_3), \ldots, (Q_1 \cup Q_2 \cup \ldots \cup Q_k)$ on machines $2, 4, \ldots, k$. After recursion, machines with even indices

$2j$ has the set $S_1 \cup S_2 \cup \ldots \cup S_{2j}$. Then, in one round, machines with odd indices $2j+1$ communicate with machine $2j$ to learn about $S_1 \cup S_2 \cup \ldots \cup S_{2j+1}$. If k is odd, we just do the same on $S_1, S_2, \ldots, S_{k-1}$ and then compute $S_1 \cup S_2 \cup \ldots \cup S_k$ in one round.

There are $O(\log k)$ recursion levels and therefore, the total number of rounds is $O(\log k)$.

Lemma 3. *Let $\mathcal{S} = \{S_1, \ldots, S_r\}$ be a collection of $r = (1+\gamma)k$ sets whose union contains L elements where $\gamma \in [0,1)$, then there exist k sets in $\mathcal{S}$ whose union contains at least $(1-\gamma)L$ elements. Furthermore, we can find these k sets in $O(\log r)$ rounds.*

Proof. Consider the following quantities $\phi_1 = |S_1|, \phi_2 = |S_1 \cup S_2| - |S_1|, \phi_3 = |S_1 \cup S_2 \cup S_3| - |S_1 \cup S_2|, \ldots$

Clearly, $\sum_{j=1}^{r} \phi_j = L$. We say S_j is responsible for element i if $i \in S_j \setminus (\bigcup_{l<j} S_l)$. This establishes a one-to-one correspondence between the sets $S_1, \ldots, S_r$ and the elements they cover. S_j is responsible for exactly ϕ_j elements. Furthermore, if we remove some sets from $\mathcal{S}$, and an element becomes uncovered, the set responsible for that element must have been removed. Thus, if we remove the γk sets corresponding to the γk smallest ϕ_j, then at most γL elements will not have a responsible set. Thus, the number of elements that become uncovered is at most γL.

To find these sets, we apply Lemma 2 with r in place of k and $O(\epsilon)$ in place of γ to learn about $\phi_1, \phi_2, \ldots, \phi_r$ in $O(\log r) = O(\log k)$ rounds. We then remove the $\gamma k = O(\epsilon)k$ sets corresponding to the γk smallest ϕ_j and output the remaining k sets.

Putting it all together. We spend $O(1/\epsilon^3 \cdot \log n \cdot \log m)$ rounds to approximately solve the linear program Π_1. From there, we can round the solution to find a sub-collection of $k + O(\epsilon)k$ sets that cover at least $(1 - 1/e - O(\epsilon))$OPT elements with high probability in $O(1/\epsilon \cdot \log m)$ rounds. We then apply Lemma 3 to find k sets among these that cover at least $(1 - 1/e - O(\epsilon))$OPT elements in $O(\log k)$ rounds. The total number of rounds is therefore $O(1/\epsilon^3 \cdot \log n \cdot \log m)$.

Reducing the number of rounds to $O(1/\epsilon^3 \cdot \log m \cdot (\log m + \log(1/\epsilon)))$. The described algorithm runs in $O(1/\epsilon^3 \cdot \log n \cdot \log m)$ rounds. Our main result in Theorem 1 states a stronger bound $O(1/\epsilon^3 \cdot \log m \cdot (\log m + \log(1/\epsilon)))$ rounds. We achieve this by adopting the sub-sampling framework of McGregor and Vu [22].

Without loss of generality, we may assume that each element is covered by some set. If not, we can remove all of the elements that are not covered by any set using $O(\log m)$ rounds. Specifically, let $\mathbf{v}_j$ be the characteristic vector of S_j. We can compute $\mathbf{v} = \sum_{i=1}^{j} \mathbf{v}_j$ in $O(\log m)$ rounds using the standard converge-cast binary tree algorithm. We can then remove the elements that are not covered by any set (elements corresponding to 0 entries in $\mathbf{v}$).

We now have m sets covering n elements. Since $k = \Omega(m)$, we must have that OPT $= \Omega(n)$. McGregor and Vu showed that if one samples each element in the universe $[n]$ independently with probability $p = \Theta\left(\frac{\log\binom{m}{k}}{\epsilon^2 \mathrm{OPT}}\right)$ then with high probability, if we run a β approximation algorithm on the subsampled universe, the solution will correpond to a $\beta - \epsilon$ approximation on the original universe. We have just argued that $\mathrm{OPT} = \Omega(n)$ and therefore with high probability, we sample $O\left(\frac{\log\binom{m}{k}}{\epsilon^2}\right) = O(1/\epsilon^2 \cdot m)$ elements by appealing to Chernoff bound and the fact that $\binom{m}{k} \leq 2^m$.

As a result, we may assume that $n = O(1/\epsilon^2 \cdot m)$. This results in an $O(1/\epsilon^2 \cdot \log m \cdot (\log m + \log(1/\epsilon)))$ round algorithm.

Bounded frequency. Assuming $f = \max_i f_i$ is known, we can lift the assumption that $k = \Omega(m)$ and parameterize our algorithm based on f instead. McGregor et al. [21] showed that the largest $\lceil kf/\eta \rceil$ sets contain a solution that covers at least $(1-\eta)$OPT elements. We therefore can assume that $m = O(kf/\eta)$ by keeping only the largest $\lceil kf/\eta \rceil$ sets which can be identified in $O(1)$ rounds.

We set $\epsilon = \eta^2/f$ and proceed to obtain a solution that covers at least $(1 - \eta^2/f)(1-\eta)\mathrm{OPT} = (1 - O(\eta))\mathrm{OPT}$ elements using at most $k + O(\epsilon m) = k + O(\eta^2/f \cdot kf/\eta) = k + O(\eta k)$ sets as in the discussion above. Appealing to Lemma 3, we can find k sets that cover at least $(1 - O(\eta))$OPT elements. The total number of rounds is $O(f^3/\eta^6 \cdot \log \frac{kf}{\eta} \cdot (\log \frac{1}{\eta} + \log \frac{kf}{\eta})) = O(f^3/\eta^6 \cdot \log^2 \frac{kf}{\eta})$.

2.2 Approximate the LP's solution via multiplicative weights

Fix an objective value L. Let P be a convex region defined by

$$P = \{(\mathbf{x}, \mathbf{z}) \in [0,1]^n \times [0,1]^m : \sum_{i \in [n]} x_i = L \text{ and } \sum_{j \in [m]} z_j = m - k\}.$$

Note that if $(\mathbf{x}_1, \mathbf{z}_1), (\mathbf{x}_2, \mathbf{z}_2), \ldots, (\mathbf{x}_T, \mathbf{z}_T) \in P$ then $\left(\frac{1}{T}\sum_{t=1}^T \mathbf{x}_t, \frac{1}{T}\sum_{t=1}^T \mathbf{z}_t\right) \in P$. Consider the following problem Ψ_1 that asks to either correctly declare that

$$\nexists (\mathbf{x}, \mathbf{z}) \in P : \frac{x_i}{f_i} + \frac{1}{f_i} \cdot \sum_{S_j \ni i} z_j \leq 1, \quad \forall i \in [n]$$

or to output a solution $(\mathbf{x}, \mathbf{z}) \in P$ such that

$$\frac{x_i}{f_i} + \frac{1}{f_i} \cdot \sum_{S_j \ni i} z_j \leq 1 + \epsilon \quad \forall i \in [n].$$

Once we have such an algorithm, we can try different values of $L = \lfloor (1+\epsilon)^0 \rfloor, \lfloor (1+\epsilon)^1 \rfloor, \lfloor (1+\epsilon)^2 \rfloor, \ldots, n$ and return the solution corresponding to the largest L that has a feasible solution. There are $O(1/\epsilon \cdot \log n)$ such guesses. We know that the guess L where $\mathrm{OPT}/(1+\epsilon) \leq L \leq \mathrm{OPT}$ must result in a feasible solution.

To avoid introducing a $\log n$ factor in the number of rounds, we partition these $O(1/\epsilon \cdot \log n)$ guesses into batches of size $O(1/\epsilon)$ where each batch corresponds to $O(\log n)$ guesses. Algorithm copies that correspond to guesses in the same batch will run in parallel. This will only introduce a $\log n$ factor in terms of memory used by each machine. By returning the solution corresponding to the largest feasible guess L, one attains Theorem 2.

Oracle implementation. Given a weight vector $\mathbf{w} \in \mathbb{R}^n$ in which $w_i \geq 0$ for all $i \in [n]$. We first consider an easier feasibility problem Ψ_2. It asks to either correctly declares that

$$\nexists (\mathbf{x}, \mathbf{z}) \in P : \sum_{i=1}^{n} w_i \cdot \left(\frac{x_i}{f_i} + \sum_{S_j \ni i} \frac{z_i}{f_i} \right) \leq \sum_{i=1}^{n} w_i, \quad \forall i \in [n]$$

or to outputs a solution $(\mathbf{x}, \mathbf{z}) \in P$ such that

$$\sum_{i=1}^{n} w_i \cdot \left(\frac{x_i}{f_i} + \sum_{S_j \ni i} \frac{z_i}{f_i} \right) \leq \sum_{i=1}^{n} w_i + \frac{1}{n^5}, \quad \forall i \in [n]. \tag{4}$$

That is, if the input is feasible, then output the corresponding $(\mathbf{x}, \mathbf{z}) \in P$ that approximately satisfy the constraint. Otherwise, correctly conclude that the input is infeasible. In the multiplicative weights update framework, this is known as the approximate oracle. Note that if there is a feasible solution to Ψ_1, then there is a feasible solution to Ψ_2 since

$$\frac{x_i}{f_i} + \frac{1}{f_i} \cdot \sum_{S_j \ni i} z_j \leq 1 \quad \forall i \in [n] \implies \sum_{i=1}^{n} w_i \cdot \left(\frac{x_i}{f_i} + \sum_{S_j \ni i} \frac{z_i}{f_i} \right) \leq \sum_{i=1}^{n} w_i.$$

We can implement an oracle that solves the above feasibility problem Ψ_2 as follows. First, observe that

$$\sum_{i=1}^{n} \frac{w_i}{f_i} \cdot \left(x_i + \sum_{S_j \ni i} z_j \right) \leq \sum_{i=1}^{n} w_i$$
$$\iff \sum_{i=1}^{n} \frac{w_i}{f_i} \cdot x_i + \sum_{j=1}^{m} z_j \cdot \sum_{i \in S_j} \frac{w_i}{f_i} \leq \sum_{i=1}^{n} w_i.$$

To ease the notation, define

$$p_i := \frac{w_i}{f_i}, \quad \forall i \in [n], \text{ and } q_j := \sum_{i \in S_j} \frac{w_i}{f_i} = \sum_{i \in S_j} p_i, \quad \forall j \in [m].$$

We therefore want to check if there exists $(\mathbf{x}, \mathbf{z}) \in P$ such that

$$LHS(\mathbf{x}, \mathbf{z}) := \sum_{i=1}^{n} x_i p_i + \sum_{j=1}^{m} z_j q_j \leq \sum_{i=1}^{n} w_i.$$

We will minimize the left hand side by minimizing each sum separately. We can indeed do this exactly. However, there is a subtle issue where we need to bound the number of bits required to represent $p_i = \frac{w_i}{f_i}$ and $q_j = \sum_{i \in S_j} \frac{w_i}{f_i} = \sum_{i \in S_j} p_i$ given the memory constraint. To do this, we truncate the value of each p_i after the $(10 \log_2 n)$-th bit following the decimal point. Note that this will result in an underestimate of p_i by at most $1/n^{10}$. In particular, let $\hat{p}_i$ be p_i after truncating the value of p_i at the $(10 \log_2 n)$-th bit after the decimal point and $\hat{q}_j = \sum_{i \in S_j} \hat{p}_i$. For any $(\mathbf{x}, \mathbf{z}) \in [0,1]^n \times [0,1]^m$, we can show that

$$\widehat{LHS}(\mathbf{x}, \mathbf{z}) := \sum_{i=1}^{n} \hat{p}_i x_i + \sum_{j=1}^{m} z_j \sum_{i \in S_j} \hat{p}_i > LHS(\mathbf{x}, \mathbf{z}) - \frac{1}{n^5}$$

Therefore, $LHS(\mathbf{x}, \mathbf{z}) - 1/n^5 \le \widehat{LHS}(\mathbf{x}, \mathbf{z}) \le LHS(\mathbf{x}, \mathbf{z})$. Note that since $\sum_{i=1}^n x_i = L$ and $\sum_{j=1}^m z_j = m - k$, to minimize $\widehat{LHS}(\mathbf{x}, \mathbf{z})$ over $(\mathbf{x}, \mathbf{z}) \in P$, we simply set

$$\begin{aligned} x_i &= [\hat{p}_i \text{ is among the } L \text{ smallest values of } \{\hat{p}_t\}_{t=1}^n], \\ z_j &= [\hat{q}_j \text{ is among the } m-k \text{ smallest values of } \{\hat{q}_t\}_{t=1}^m]. \end{aligned}$$

After setting $\mathbf{x}, \mathbf{z}$ as above, if $\widehat{LHS}(\mathbf{x}, \mathbf{z}) > \sum_{i=1}^n w_i \implies LHS(\mathbf{x}, \mathbf{z}) > \sum_{i=1}^n w_i$, then it is safe to declare that the system is infeasible. Otherwise, we have found $(\mathbf{x}, \mathbf{z}) \in P$ such that $\widehat{LHS}(\mathbf{x}, \mathbf{z}) \le \sum_{i=1}^n w_i \implies LHS(\mathbf{x}, \mathbf{z}) \le \sum_{i=1}^n w_i + 1/n^5$ as required by Equation (4).

Lemma 4. *Assume that all machines have the vector $\boldsymbol{w}$. We can solve the feasibility problem Ψ_2 in $O(1)$ rounds, where all machines either learn that the system is infeasible or obtain an approximate solution $(\boldsymbol{x}, \boldsymbol{z}) \in P$ that satisfies Equation (4).*

Solving the LP via multiplicative weights. Once the existence of such an oracle is guaranteed, we can follow the multiplicative weights framework to approximately solve the LP. We will first explain how to implement the MWU algorithm in the MPC model. See Algorithm 2.

Lemma 5. *Algorithm 2 can be implemented in $O(1/\epsilon^2 \cdot \log n \cdot \log m)$ rounds.*

The next lemma is an adaptation of the standard multiplicative weights algorithm.

Lemma 6. *The output of Algorithm 2 satisfies the following property. If there exists a feasible solution, then the output satisfies:*

$$\frac{x_i}{f_i} + \sum_{S_j \ni i} \frac{z_j}{f_i} \le 1 + \epsilon \quad \forall i \in [n], \text{ and } \sum_{i=1}^{n} x_i = L.$$

Otherwise, the algorithm correctly concludes that the system is infeasible.

Algorithm 2: Multiplicative weights for solving the LP

```
   Input: Objective value $L$, $\epsilon \leq 1/4$
1  Initialize $w_i^{(0)} = 1$ for all $i \in [n]$.
2  for iteration $t = 1, 2, \dots, T = O(1/\epsilon^2 \cdot \log n)$ do
3      Run the oracle in Lemma 4 with $\mathbf{w}^{(t-1)}$ to check if there exists a feasible
         solution. If the answer is INFEASIBLE, stop the algorithm. If the answer is
         FEASIBLE, let $\mathbf{x}^{(t)}$ and $\mathbf{z}^{(t)}$ be the output of the oracle that are now
         stored in all machines .
4      Each machine $j$ constructs $Y_j = \{Y_{j1}, Y_{j2}, \dots, Y_{jn}\}$ where
         $Y_{ji} = z_j^{(t)} \cdot [\{i \in S_j\}]$.
5      Compute $W = \sum_{j \in [m]} Y_j$ in $O(\log m)$ rounds using the a converge-cast
         binary tree and send $W$ to the central machine. Note that
         $W_i = \sum_{S_j \ni i} z_j^{(t)}$.
6      For each $i \in [n]$, the central machine computes $E_i^{(t)} = f_i \cdot \text{error}_i^{(t)}$ where
         $\text{error}_i^{(t)} := 1 - \frac{x_i^{(t)}}{f_i} - \frac{W_i}{f_i} = 1 - \frac{x_i^{(t)}}{f_i} - \sum_{S_j \ni i} \frac{z_j^{(t)}}{f_i} \quad \forall i \in [n]$ and sends
         $\sum_{d=1}^{t} E_i^{(d)}$ to all other machines.
7      For each $i \in [n]$, each machine computes
         $w_i^{(t)} = 2^{-\epsilon \cdot \sum_{d=1}^{t} E_i^{(d)}/f_i} = 2^{-\epsilon \cdot \sum_{d=1}^{t} \text{error}_i^{(d)}} = 2^{-\epsilon \cdot \text{error}_i^{(t)}} w_i^{(t-1)}$.
8  After $T$ iterations, output $\mathbf{x} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{x}^{(t)}$ and $\mathbf{z} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{z}^{(t)}$.
```

Proof (sketch). If the algorithm does not output INFEASIBLE, this implies that in each iteration t, $\sum_{i=1}^{n} x_i^{(t)} = L$ and $\sum_{j=1}^{m} z_j^{(t)} = m - k$. Hence, the output $(\mathbf{x}, \mathbf{z}) \in P$. Define the potential function $\Phi^{(t)} := \sum_{i=1}^{n} w_i^{(t)}$.

We will make use of the fact $\exp(-\eta x) \leq 1 - \eta x + \eta^2 x^2$ for $|\eta x| \leq 1$. Note that for all i and t, we always have $\text{error}_i^{(t)} = 1 - \frac{x_i^{(t)}}{f_i} - \sum_{S_j \ni i} \frac{z_j^{(t)}}{f_i} \in [-1, 1]$. Let $\alpha = \epsilon \cdot \ln(2)$, as long as $\epsilon \leq 1/4$, we have $|\alpha \cdot \text{error}_i^{(t)}| < 1$ and therefore

$$w_i^{(t)} = \exp\left(-\alpha \cdot \text{error}_i^{(t)}\right) \cdot w_i^{(t-1)} \leq \left(1 - \alpha \cdot \text{error}_i^{(t)} + \alpha^2 \cdot \left(\text{error}_i^{(t)}\right)^2\right) \cdot w_i^{(t-1)}.$$

Summing over i gives:

$$\begin{aligned} \Phi^{(t)} &\leq \sum_{i=1}^{n} \left(1 - \alpha \cdot \text{error}_i^{(t)} + \alpha^2\right) \cdot w_i^{(t-1)} \\ &= (1 + \alpha^2) \sum_{i=1}^{n} w_i^{(t-1)} - \alpha \sum_{i=1}^{n} \text{error}_i^{(t)} \cdot w_i^{(t-1)}. \end{aligned}$$

The first inequality follows from the fact that $\left(\text{error}_i^{(t)}\right)^2 \in [0, 1]$. Note that

$$\sum_{i=1}^{n} \text{error}_i^{(t)} w_i^{(t-1)} = \sum_{i=1}^{n} w_i^{(t-1)} \left(1 - \frac{x_i^{(t-1)}}{f_i} - \sum_{S_j \ni i} \frac{z_j^{(t-1)}}{f_i}\right)$$

$$= \sum_{i=1}^{n} w_i^{(t-1)} - \sum_{i=1}^{n} w_i^{(t-1)} \left(\frac{x_i^{(t-1)}}{f_i} + \sum_{S_j \ni i} \frac{z_j^{(t-1)}}{f_i} \right) \geq -\frac{1}{n^5}.$$

The last inequality follows from the oracle's guarantee. Thus, $\Phi^{(t)} \leq (1 + \alpha^2) \sum_{i=1}^{n} w_i^{(t-1)} + \frac{\alpha}{n^5} = (1+\alpha^2)\Phi^{t-1} + \frac{\alpha}{n^5}$. We can show by induction that

$$\Phi^{(T)} \leq (1+\alpha^2)^T \Phi^{(0)} + \frac{1}{\alpha n^5}(1+\alpha^2)^T.$$

Recall that $\alpha = \epsilon \ln 2$ and we assume $1/\epsilon < n/10$. Thus, $1/(\alpha n^5) < \epsilon^4/\ln 2 < 1$. Furthermore, recall that $\Phi^{(0)} = n$. We have,

$$\begin{aligned} w_i^{(T)} &\leq (1+\alpha^2)^T(n+1) < (1+\alpha^2)^T 2n \\ \exp\left(-\alpha \sum_{t=1}^{T} \text{error}_i^{(t)}\right) &\leq (1+\alpha^2)^T(2n) \\ -\alpha \sum_{t=1}^{T} \text{error}_i^{(t)} &\leq \ln(2n) + T\ln(1+\alpha^2). \end{aligned}$$

We use the fact that $\ln(1+x) \leq x$ for $x \in \mathbb{R}$ to get

$$\begin{aligned} \sum_{t=1}^{T} \text{error}_i^{(t)} &\geq -\frac{\ln(2n)}{\alpha} - T\frac{\ln(1+\alpha^2)}{\alpha} \\ \sum_{t=1}^{T} \left(1 - \frac{x_i^{(t)}}{f_i} - \sum_{S_j \ni i} \frac{z_j^{(t)}}{f_i}\right) &\geq -\frac{\ln(2n)}{\alpha} - T\frac{\alpha^2}{\alpha} \\ \frac{1}{T}\sum_{t=1}^{T} \left(1 - \frac{x_i^{(t)}}{f_i} - \sum_{S_j \ni i} \frac{z_j^{(t)}}{f_i}\right) &\geq -\frac{\ln(2n)}{T\alpha} - \alpha \\ 1 - \frac{x_i}{f_i} - \sum_{S_j \ni i} \frac{z_j}{f_i} &\geq -\frac{\ln(2n)}{T\alpha} - \alpha \\ \frac{x_i}{f_i} + \sum_{S_j \ni i} \frac{z_j}{f_i} \leq 1 + \frac{\ln(2n)}{T\alpha} + \alpha &\implies \frac{x_i}{f_i} + \sum_{S_j \ni i} \frac{z_j}{f_i} \quad \leq 1 + O(\epsilon). \end{aligned}$$

The last inequality follows from choosing $T = \Theta(1/\epsilon^2 \cdot \log n)$ and the fact that $\alpha = \epsilon \ln(2)$; furthermore, recall that the final solution $x_i = \frac{1}{T}\sum_{t=1}^{T} x_i^{(t)}$ and $z_j = \frac{1}{T}\sum_{t=1}^{T} z_j^{(t)}$. Thus, the output of the algorithm satisfies the desired properties.

References

1. Anagnostopoulos, A., Becchetti, L., Bordino, I., Leonardi, S., Mele, I., Sankowski, P.: Stochastic query covering for fast approximate document retrieval. ACM Trans. Inf. Syst. **33**(3), 11:1–11:35 (2015)

2. Arora, S., Hazan, E., Kale, S.: The multiplicative weights update method: a meta-algorithm and applications. Theory Comput. **8**(1), 121–164 (2012)
3. Assadi, S.: Tight space-approximation tradeoff for the multi-pass streaming set cover problem. In: PODS. pp. 321–335. ACM (2017)
4. Assadi, S., Khanna, S.: Tight bounds on the round complexity of the distributed maximum coverage problem. In: SODA. pp. 2412–2431. SIAM (2018)
5. Assadi, S., Khanna, S., Li, Y.: Tight bounds for single-pass streaming complexity of the set cover problem. SIAM J. Comput. **50**(3) (2021)
6. Cervenjak, P., Gan, J., Umboh, S.W., Wirth, A.: Maximum unique coverage on streams: Improved FPT approximation scheme and tighter space lower bound. In: APPROX/RANDOM. LIPIcs, vol. 317, pp. 25:1–25:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024)
7. Chakrabarti, A., McGregor, A., Wirth, A.: Improved algorithms for maximum coverage in dynamic and random order streams. CoRR **abs/2403.14087** (2024)
8. Feige, U.: A threshold of ln n for approximating set cover. J. ACM **45**(4), 634–652 (1998)
9. Har-Peled, S., Indyk, P., Mahabadi, S., Vakilian, A.: Towards tight bounds for the streaming set cover problem. In: PODS. pp. 371–383. ACM (2016)
10. Hochbaum, D.S., Pathria, A.: Analysis of the greedy approach in problems of maximum k-coverage. Naval Research Logistics (NRL) **45**(6), 615–627 (1998)
11. Indyk, P., Mahabadi, S., Rubinfeld, R., Ullman, J.R., Vakilian, A., Yodpinyanee, A.: Fractional set cover in the streaming model. In: APPROX-RANDOM. LIPIcs, vol. 81, pp. 12:1–12:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
12. Indyk, P., Vakilian, A.: Tight trade-offs for the maximum k-coverage problem in the general streaming model. In: PODS. pp. 200–217. ACM (2019)
13. Jaud, S., Wirth, A., Choudhury, F.M.: Maximum coverage in sublinear space, faster. In: SEA. LIPIcs, vol. 265, pp. 21:1–21:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023)
14. Karloff, H.J., Suri, S., Vassilvitskii, S.: A model of computation for mapreduce. In: SODA. pp. 938–948. SIAM (2010)
15. Kempe, D., Kleinberg, J.M., Tardos, É.: Maximizing the spread of influence through a social network. Theory Comput. **11**, 105–147 (2015)
16. Khanna, S., Konrad, C., Alexandru, C.: Set cover in the one-pass edge-arrival streaming model. In: PODS. pp. 127–139. ACM (2023)
17. Krause, A., Guestrin, C.: Near-optimal observation selection using submodular functions. In: AAAI. pp. 1650–1654. AAAI Press (2007)
18. Kumar, R., Moseley, B., Vassilvitskii, S., Vattani, A.: Fast greedy algorithms in mapreduce and streaming. ACM Trans. Parallel Comput. **2**(3), 14:1–14:22 (2015)
19. Ladner, R.E., Fischer, M.J.: Parallel prefix computation. J. ACM **27**(4), 831–838 (1980)
20. Liu, P., Vondrák, J.: Submodular optimization in the mapreduce model. In: SOSA. OASIcs, vol. 69, pp. 18:1–18:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
21. McGregor, A., Tench, D., Vu, H.T.: Maximum coverage in the data stream model: Parameterized and generalized. In: ICDT. LIPIcs, vol. 186, pp. 12:1–12:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
22. McGregor, A., Vu, H.T.: Better streaming algorithms for the maximum coverage problem. Theory Comput. Syst. **63**(7), 1595–1619 (2019)
23. da Ponte Barbosa, R., Ene, A., Nguyen, H.L., Ward, J.: A new framework for distributed submodular maximization. In: FOCS. pp. 645–654. IEEE Computer Society (2016)

24. Saha, B., Getoor, L.: On maximum coverage in the streaming model & application to multi-topic blog-watch. In: SDM. pp. 697–708. SIAM (2009)
25. Warneke, R., Choudhury, F.M., Wirth, A.: Maximum coverage in random-arrival streams. In: ESA. LIPIcs, vol. 274, pp. 102:1–102:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023)