

The Online Set Cover Problem

Noga Alon* Baruch Awerbuch† Yossi Azar‡ Niv Buchbinder § Joseph (Seffi) Naor ¶

Abstract

Let $X = \{1, 2, \dots, n\}$ be a ground set of n elements, and let \mathcal{S} be a family of subsets of X , $|\mathcal{S}| = m$, with a positive cost c_S associated with each $S \in \mathcal{S}$.

Consider the following online version of the set cover problem, described as a game between an algorithm and an adversary. An adversary gives elements to the algorithm from X one-by-one. Once a new element is given, the algorithm has to cover it by some set of \mathcal{S} containing it. We assume that the elements of X and the members of \mathcal{S} are known in advance to the algorithm, however, the set $X' \subseteq X$ of elements given by the adversary is not known in advance to the algorithm. (In general, X' may be a strict subset of X .) The objective is to minimize the total cost of the sets chosen by the algorithm. Let \mathcal{C} denote the family of sets in \mathcal{S} that the algorithm chooses. At the end of the game the adversary also produces (off-line) a family of sets \mathcal{C}_{OPT} that covers X' . The performance of the algorithm is the ratio between the cost of \mathcal{C} and the cost of \mathcal{C}_{OPT} . The maximum ratio, taken over all input sequences, is the *competitive ratio* of the algorithm.

We present an $O(\log m \log n)$ competitive deterministic algorithm for the problem, and establish a nearly matching $\Omega\left(\frac{\log n \log m}{\log \log m + \log \log n}\right)$ lower bound for all interesting values of m and n . The techniques used are motivated by similar techniques developed in computational learning theory for online prediction (e.g., the WINNOW algorithm) together with a novel way of converting the fractional solution they supply into a deterministic online algorithm.

1 Introduction

The *set cover* problem is defined as follows. Let $X = \{1, 2, \dots, n\}$ be a ground set of n elements, and let \mathcal{S} be a family of subsets of X , $|\mathcal{S}| = m$. A *cover* is a collection of sets such that their union is X . Each $S \in \mathcal{S}$ has a non-negative *cost* $c(S)$ associated with it. The goal is to find a cover of

*Schools of Mathematics and Computer Science, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv, Israel. Email: nogaa@post.tau.ac.il. Research supported in part by a US-Israel BSF grant, by the Israel Science Foundation and by the Hermann Minkowski Minerva Center for Geometry at Tel Aviv University.

†Johns Hopkins University, Baltimore, MD 21218. E-mail: baruch@blaze.cs.jhu.edu. Supported by Air Force Contract TNDGAFOSR-86-0078, ARPA/Army contract DABT63-93-C-0038, ARO contract DAAL03-86-K-0171, NSF contract 9114440-CCR, DARPA contract N00014-J-92-1799, and a special grant from IBM.

‡School of Computer Science, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv, Israel. Email: azar@post.tau.ac.il. Research supported in part by the Israel Science Foundation and by the IST Program of the EU.

§Computer Science Dept., Technion, Haifa 32000, Israel. E-mail: nivb@cs.technion.ac.il.

¶Computer Science Dept., Technion, Haifa 32000, Israel. E-mail: naor@cs.technion.ac.il. Research supported in part by EU contract IST-1999-14084 (APPOL II).

minimum cost. The set cover problem is a classic NP-hard problem that was studied extensively in the literature, and the best approximation factor achievable for it in polynomial time (assuming $P \neq NP$) is $\Theta(\log n)$ [6, 7, 9, 10].

Consider the following online version of the set cover problem, described as a game between an algorithm and an adversary. An adversary gives elements to the algorithm from X one-by-one. Once a new element is given, the algorithm has to cover it by some set of \mathcal{S} containing it. Denote by $X' \subseteq X$ the set of elements given by the adversary. Our assumption is that the set cover instance, i.e. the elements of X and the members of \mathcal{S} , is known in advance to the algorithm. The objective is to minimize the total cost of the sets chosen by the algorithm. However, the algorithm does not know in advance the set of elements X' given by the adversary, i.e., X' may be a strict subset of X in general. Let \mathcal{C} denote the family of sets in \mathcal{S} that the algorithm chooses. At the end of the game the adversary also produces (off-line) a family of sets \mathcal{C}_{OPT} that covers all the elements belonging to X' . The performance of the algorithm is defined to be the ratio between the cost of \mathcal{C} and the cost of \mathcal{C}_{OPT} . The maximum ratio, taken over all input sequences, is defined to be the *competitive ratio* of the algorithm.

The online set cover problem captures many practical scenarios. Consider, for example, servers in a network that provide a service. There is a set of potential clients that may need the service and each server can provide the service to a subset of them. (E.g., the subset is determined by the distance from the server.) There is a setup cost, or activation cost, associated with the operation of a server. The clients arrive one-by-one. Upon arrival of a client, the network manager has to decide which server to activate so that the client receives the service it requested. The network manager knows in advance the set of potential clients and the set of servers, however, it does not know in advance which clients will indeed request the service.

1.1 Results

Our main result is an $O(\log m \log n)$ competitive algorithm for the online set cover problem. We first present the algorithm for the unweighted case, i.e., when all sets have unit cost. Then, we generalize the algorithm for the weighted case, achieving the same competitive factor. If each element appears in at most d sets, and all sets have unit cost, then the competitive factor of our algorithm can be improved to $O(\log d \log n)$.

The algorithm associates a weight with each set, initially all weights are equal. In each iteration of the algorithm, when the adversary gives a new element, all the sets containing the element multiply their weight by a factor (which depends on the cost of the set, among other parameters). A set chooses itself to the solution with probability roughly proportional to the increase in its weight. We define a potential function that depends on the weight of the uncovered elements, the cost of the sets already in the cover, and the cost and weight of the sets not belonging to the cover. The heart of our analysis is the claim that there exists a choice of sets in each iteration for which the potential function is non-increasing. This is proved by analyzing a suitable randomized choice of sets. We then show that the randomized choice of the sets in each iteration can be derandomized, thus making the online algorithm deterministic, while maintaining the same competitive factor. We note that knowing in advance the set cover instance is crucial for making the online algorithm deterministic.

A high level description of the design of the algorithm is as follows. It starts by producing online

a fractional solution to the problem, where the fractional solution (at least for the unweighted case) is motivated by similar techniques developed in computational learning theory for online prediction [11, 8] (e.g., the WINNOW algorithm). See also [5, 3, 4] for related techniques and additional references. Fractional solutions can often be converted into randomized algorithms, but it is usually impossible to perform this conversion online. In our case, however, this conversion is possible, because of the way the fractional solution evolves in time. Finally, the randomized algorithm is converted into a deterministic one by using an appropriate derandomization technique. This derandomization is non-standard, as it has to apply to the online setting. The requirement to maintain the properties of the online solution obtained leads to the potential function used.

Our result is nearly tight. We prove a lower bound of $\Omega\left(\frac{\log n \log m}{\log \log m + \log \log n}\right)$ on the competitiveness of any deterministic algorithm for the online set cover problem for a wide range of the parameters m and n , and observe that this range cannot be extended significantly. Thus, our upper and lower bounds almost match for all interesting values of the parameters.

We note that the problem considered here is different from the online set cover problem discussed in [2]. There, we are also given m sets and n elements that arrive one at a time. However, the goal of the online algorithm is to pick k sets so as to maximize the number of elements that are covered. The algorithm only gets credit for elements that are contained in a set that it selected *before* or *during* the step in which the element arrived. The authors of [2] showed a *randomized* $\Theta(\log m \log \frac{n}{k})$ competitive algorithm for the problem, where the bound is optimal for many values of n , m , and k . This problem is different from our problem here, as it deals with maximum benefit, whereas we consider minimum cost. Indeed, it is easy to see that in contrast to our case, the problem in [2] does not yield any non-trivial deterministic algorithm, and the algorithms and techniques for the two problems seem to be totally unrelated, despite the similarity in the description of the two problems.

2 The Unweighted Case

We describe in this section an $O(\log m \log n)$ competitive algorithm for the unweighted case, i.e., when all sets have unit cost.

The algorithm maintains a weight $w_S > 0$ for each $S \in \mathcal{S}$. The weights can only increase during the run of the algorithm. Initially, $w_S = 1/(2m)$ for each $S \in \mathcal{S}$. The weight of each element $j \in X$ is defined as $w_j = \sum_{S \in \mathcal{S}_j} w_S$, where \mathcal{S}_j denotes the collection of sets containing element j . Initially, the algorithm starts with the empty cover $\mathcal{C} = \emptyset$. Define C to be the set of all elements covered by the members of \mathcal{C} . (Initially, $C = \emptyset$.) The following potential function is used throughout the algorithm:

$$\Phi = \sum_{j \notin C} n^{2w_j}$$

We now give a high level description of a single iteration of the algorithm in which the adversary gives an element j and the algorithm chooses sets that cover it.

1. If $w_j \geq 1$, then do not add any new set to \mathcal{C} and do not update the weights of the sets.
2. Else, ($w_j < 1$), perform a *weight augmentation*:

- (a) Let k be the minimal integer for which $2^k \cdot w_j > 1$. (Clearly, $2^k \cdot w_j < 2$.)
- (b) For each set $S \in \mathcal{S}_j$, $w_S \leftarrow 2^k \cdot w_S$.
- (c) Choose from \mathcal{S}_j at most $4 \log n$ sets to \mathcal{C} so that the value of the potential function Φ does not exceed its value before the weight augmentation.

In the following we analyze the performance of the algorithm and explain which sets to add to the cover \mathcal{C} in the weight augmentation step.

Lemma 1. *The total number of iterations in which a weight augmentation step is performed is at most $|\mathcal{C}_{OPT}| \cdot (\log m + 2)$.*

Proof. For each subset S , $w_S \leq 2$ always holds, since the algorithm maintains in all iterations that $w_j \leq 2$ for all elements j . Consider an iteration in which the adversary gives element j . A weight augmentation is performed in this iteration if and only if $w_j < 1$. When doing a weight augmentation, the weight of at least one set belonging to \mathcal{C}_{OPT} is multiplied by a factor greater than or equal to two. Since the weight of each set is initially $1/(2m)$ and at the end at most 2, it follows that each set can participate in at most $\log(4m)$ iterations in which a weight augmentation is performed. Hence, the desired result follows. \square

Lemma 2. *Consider an iteration in which a weight augmentation is performed. Let Φ_s and Φ_e be the values of the potential function Φ before and after the iteration, respectively. Then, there exist at most $4 \log n$ sets that can be added to \mathcal{C} during the iteration such that $\Phi_e \leq \Phi_s$.*

Proof. The proof is probabilistic. Suppose that the adversary gives element j in the iteration. For each set $S \in \mathcal{S}_j$, let w_S and $w_S + \delta_S$ denote the weight of S before and after the iteration, respectively. Define $\delta_j = \sum_{S \in \mathcal{S}_j} \delta_S$. The algorithm maintains that $w_j + \delta_j = \sum_{S \in \mathcal{S}_j} (w_S + \delta_S) \leq 2$.

We now explain which sets from \mathcal{S}_j are added to \mathcal{C} .

Repeat $4 \log n$ times:

choose at most one set from \mathcal{S}_j such that each set $S \in \mathcal{S}_j$ is chosen with probability $\delta_S/2$. (This can be implemented by choosing a number uniformly at random in $[0, 1]$, since $\delta_j/2 \leq 1$.)

Consider an element $j' \in X$ such that $j' \notin \mathcal{C}$. Let the weight of j' before the iteration be $w_{j'}$ and let the weight after the iteration be $w_{j'} + \delta_{j'}$. Element j' contributes before the iteration to the potential function the value $n^{2w_{j'}}$. In each random choice, the probability that we do not choose a set containing element j' is $1 - \frac{\delta_{j'}}{2}$. The probability that this happens in all the $4 \log n$ random choices is therefore $(1 - \frac{\delta_{j'}}{2})^{4 \log n} \leq n^{-2\delta_{j'}}$.

Therefore, the expected contribution of element j' to the potential function after the iteration is at most

$$n^{-2\delta_{j'}} n^{2(w_{j'} + \delta_{j'})} = n^{2w_{j'}}.$$

By linearity of expectation it follows that $\mathbf{Exp}[\Phi_e] \leq \Phi_s$. Hence, there exists a choice of at most $4 \log n$ sets such that $\Phi_e \leq \Phi_s$. \square

Theorem 3. *At the end of the algorithm, \mathcal{C} is a feasible cover of X^t and $|\mathcal{C}|$ is $O(|\mathcal{C}_{OPT}| \log m \log n)$.*

Proof. Initially, the value of the potential function Φ is less than $n \cdot n = n^2$. It follows from Lemma 2 that Φ is non-increasing throughout the iterations. Therefore, if $w_j \geq 1$ for an element j during the algorithm, then $j \in \mathcal{C}$, otherwise $\Phi \geq n^{2w_j} \geq n^2$. Hence, \mathcal{C} is a feasible cover. It follows from Lemma 1 that the number of iterations is at most $|\mathcal{C}_{OPT}| \cdot (\log m + 2)$. By Lemma 2, in each iteration we choose at most $4 \log n$ sets to \mathcal{C} . Therefore, the total number of sets chosen by the algorithm is as claimed. \square

Remark: If every element appears in at most d sets, then the algorithm can be modified by starting with the weights $w_S = 1/(2d)$ for each $S \in \mathcal{S}$, and the competitive factor can be improved in this case to $O(\log d \log n)$.

3 The Weighted Case

We describe in this section an $O(\log m \log n)$ competitive algorithm for the weighted case. For each set $S \in \mathcal{S}$, a positive cost c_S is associated with the set. The cost of the optimal solution, $c(\mathcal{C}_{OPT})$, is denoted by α .

Note, first, that we may assume, by doubling, that the value of α is known up to a factor of 2. Indeed, we can start guessing $\alpha = \min_{S \in \mathcal{S}} c_S$, and run the algorithm with this value of the optimal solution. If it turns out that the value of the optimal solution is already at least twice our current guess for it, (that is, the cost of \mathcal{C} exceeds $\Theta(\alpha \log m \log n)$), then we “forget” about all sets chosen so far to \mathcal{C} , update the value of α by doubling it, and continue on. We note that the cost of the sets that we have “forgotten” about can increase the cost of our solution by at most a factor of 2, since the value of α was doubled in each step.

We thus assume that α is known. Hence, we can ignore all sets of cost exceeding α , and also choose all sets of cost at most α/m to \mathcal{C} . Thus, we assume that all costs are between α/m and α , and further normalize the costs so that the minimum cost is 1 (and hence the maximum cost is at most m).

We now describe an online algorithm with competitive factor $(6 + o(1)) \log m \log n$, (assuming that α is known), where the $o(1)$ term tends to zero as n and m tend to infinity. It is worth noting that the constant $6 + o(1)$ can be improved to $2 + o(1)$ by being somewhat more careful, but we prefer to describe the algorithm with the inferior constant, to simplify the computation. From now on, all $o(1)$ terms denote terms that tend to zero as n and m tend to infinity. All logarithms are in the natural basis e .

As in the unweighted case, the algorithm maintains a weight $w_S > 0$ for each $S \in \mathcal{S}$. The weights can only increase during the run of the algorithm. Initially $w_S = 1/m^2$ for each $S \in \mathcal{S}$. The weight of each element $j \in X$ is defined as $w_j = \sum_{S \in \mathcal{S}_j} w_S$, where \mathcal{S}_j denotes the collection of sets containing element j .

Initially, the algorithm starts with the empty cover $\mathcal{C} = \emptyset$. Define C to be the set of all elements

covered by the members of \mathcal{C} . The following potential function is used throughout the algorithm:

$$\Phi = \sum_{j \notin \mathcal{C}} n^{2w_j} + n \cdot \exp \left(\frac{1}{2\alpha} \sum_{S \in \mathcal{S}} (c_S \chi_{\mathcal{C}}(S) - 3w_S c_S \log n) \right).$$

The function $\chi_{\mathcal{C}}$ above is the characteristic function of \mathcal{C} , that is, $\chi_{\mathcal{C}}(S) = 1$ if $S \in \mathcal{C}$, and $\chi_{\mathcal{C}}(S) = 0$ otherwise.

We now give a high level description of a single iteration of the algorithm in which the adversary gives an element j and the algorithm chooses sets that cover it.

1. If $w_j \geq 1$, then do not add any new set to \mathcal{C} and do not update the weights of the sets.
2. Else, ($w_j < 1$), perform a sequence of *weight augmentation* steps as long as $w_j < 1$:
 - (a) For each $S \in \mathcal{S}_j$, $w_S \leftarrow w_S \cdot (1 + \frac{1}{n \cdot c_S})$
 - (b) Choose from \mathcal{S}_j sets to \mathcal{C} so that the value of the potential function Φ does not exceed its value before the weight augmentation.

In the following we analyze the performance of the algorithm and explain which sets to add to the cover \mathcal{C} in the weight augmentation step.

Lemma 4. *The total number of weight augmentation steps performed during the algorithm is at most*

$$\sum_{S \in \mathcal{C}_{OPT}} (n \cdot c_S + 1) \log \left(m^2 \left(1 + \frac{1}{n} \right) \right) \leq (2 + o(1)) n \alpha \log m.$$

Proof. Obviously, for each subset S , $w_S \leq 1 + \frac{1}{n \cdot c_S}$ always holds. Consider an iteration in which the adversary gives element j . A weight augmentation is performed in this iteration as long as $w_j < 1$. When doing a weight augmentation, the weight of at least one set $S \in \mathcal{C}_{OPT}$ is multiplied by a factor of $(1 + \frac{1}{n \cdot c_S})$. Since the weight of each set is initially $1/m^2$ and at the end at most $(1 + 1/n)$, it follows that each set S participates in at most $(n \cdot c_S + 1) \log \left(m^2 \left(1 + \frac{1}{n} \right) \right)$ steps in which a weight augmentation is performed. Hence, the desired result follows. \square

Lemma 5. *The following is maintained throughout the algorithm:*

$$\sum_{S \in \mathcal{S}} w_S c_S \leq (2 + o(1)) \alpha \log m.$$

Proof. Consider an iteration in which the adversary gives element j . We start with weights satisfying

$$\sum_{S \in \mathcal{S}_j} w_S \leq 1,$$

and increase the weight of each set S in \mathcal{S}_j by $w_S / (n \cdot c_S)$ in each step. Thus, the total increase of the quantity $\sum_{S \in \mathcal{S}} w_S c_S$ in a step does not exceed

$$\sum_{S \in \mathcal{S}_j} \frac{w_S}{n \cdot c_S} c_S = \sum_{S \in \mathcal{S}_j} \frac{w_S}{n} \leq \frac{1}{n}.$$

Initially, $\sum_{S \in \mathcal{S}} w_S c_S \leq m \cdot \frac{1}{m^2} \cdot m = 1$, and the result thus follows from Lemma 4 that bounds the number of weight augmentation steps. \square

Lemma 6. *Consider a step in which a weight augmentation is performed. Let Φ_s and Φ_e be the values of the potential function Φ before and after the step, respectively. Then, there exist sets that can be added to \mathcal{C} during the step such that $\Phi_e \leq \Phi_s$.*

Proof. The proof is probabilistic. Suppose that the adversary gives element j in the iteration. For each set $S \in \mathcal{S}_j$, let w_S and $w_S + \delta_S$ denote the weight of S before and after the step, respectively. Define $\delta_j = \sum_{S \in \mathcal{S}_j} \delta_S$.

We now explain which sets from \mathcal{S}_j are added to \mathcal{C} . Independently, for each $S \in \mathcal{S}_j$, set S is added to \mathcal{C} with probability $1 - n^{-2\delta_S}$. (This is roughly the same as choosing S with probability $\delta_S/2$ and repeating this $4 \log n$ times.) Let \mathcal{C}' denote the cover obtained from \mathcal{C} by adding to it the randomly chosen sets.

We first bound the expected value of the first term of the potential function. This is similar to the unweighted case. Consider an element $j' \in X$ such that $j' \notin \mathcal{C}$. Let the weight of j' before the step be $w_{j'}$ and let the weight after the step be $w_{j'} + \delta_{j'}$. Element j' contributes before the step to the first term of the potential function the value $n^{2w_{j'}}$. The probability that we do not choose a set containing element j' is $n^{-2\delta_{j'}}$. Therefore, the expected contribution of element j' to the potential function after the step is at most $n^{-2\delta_{j'}} n^{2(w_{j'} + \delta_{j'})} = n^{2w_{j'}}$. By linearity of expectation it follows that the expected value of $\sum_{j' \notin \mathcal{C}} n^{2w_{j'}}$ after the step is precisely its value before the step.

It remains to bound the expected value of the second term of the potential function. Let

$$T = n \cdot \exp \left(\frac{1}{2\alpha} \sum_{S \in \mathcal{S}} (c_S \chi_{\mathcal{C}}(S) - 3w_S c_S \log n) \right)$$

denote the value of the second term of the potential function before the step, and let T' denote the same term with respect to the cover \mathcal{C}' . Since the choices of different sets are independent, and the random variable T' can be viewed as a product of independent random variables, its expected value is the product of the corresponding expected values. Therefore,

$$\begin{aligned} \mathbf{Exp}[T'] &= n \cdot \exp \left(-\frac{1}{2\alpha} \sum_{S \in \mathcal{S}} 3(w_S + \delta_S) c_S \log n \right) \\ &\quad \cdot \prod_{S \in \mathcal{S}} \mathbf{Exp} \left[\exp \left(\frac{1}{2\alpha} c_S \chi_{\mathcal{C}'}(S) \right) \right] \end{aligned} \quad (1)$$

Fix an $S \in \mathcal{S}$. If the weight of S was not changed during the step, or if it was changed even though $S \in \mathcal{C}$ (i.e., j is covered, but $w_j < 1$), then the expected value of $\exp(\frac{1}{2\alpha} \cdot c_S \chi_{\mathcal{C}'}(S))$ is precisely its value before the step. Therefore, if we let \mathcal{S}' denote the family of all sets $S \in \mathcal{S} \setminus \mathcal{C}$ whose weights were changed during the weight augmentation step, then the expected value in (1) is precisely

$$\begin{aligned} \mathbf{Exp}[T'] &= T \cdot \exp \left(-\frac{1}{2\alpha} \sum_{S \in \mathcal{S}} 3\delta_S c_S \log n \right) \\ &\quad \cdot \prod_{S \in \mathcal{S}'} \mathbf{Exp} \left[\exp \left(\frac{1}{2\alpha} c_S \chi_{\mathcal{C}'}(S) \right) \right]. \end{aligned} \quad (2)$$

Consider, now, an $S \in \mathcal{S}'$. S did not belong to \mathcal{C} before the step, and after the step, the probability that $\chi_{\mathcal{C}'}(S) = 1$ is $1 - n^{-2\delta_S}$. Thus,

$$\mathbf{Exp} \left[\exp \left(\frac{1}{2\alpha} c_S \chi_{\mathcal{C}'}(S) \right) \right] \quad (3)$$

$$= n^{-2\delta_S} + (1 - n^{-2\delta_S}) \cdot \exp \left(\frac{c_S}{2\alpha} \right) \quad (4)$$

$$\leq 1 - 2\delta_S \log n + 2\delta_S \log n \exp \left(\frac{c_S}{2\alpha} \right) \quad (5)$$

$$= 1 + 2\delta_S \log n \left(\exp \left(\frac{c_S}{2\alpha} \right) - 1 \right) \quad (6)$$

$$\leq 1 + 2\delta_S \log n \frac{3c_S}{4\alpha} \quad (7)$$

$$\leq \exp \left(\frac{3\delta_S c_S \log n}{2\alpha} \right). \quad (8)$$

Here, (5) follows since for all $x \geq 0$ and $z \geq 1$, $e^{-x} + (1 - e^{-x}) \cdot z \leq 1 - x + x \cdot z$, (7) follows since $e^y - 1 \leq 3y/2$ for all $0 \leq y \leq 1/2$, and (8) follows since $1 + x \leq e^x$ for all $x \geq 0$. Plugging in (2), we conclude that the expected value of the second term after the step and random choices is at most

$$\begin{aligned} \mathbf{Exp}[T'] &= T \cdot \exp \left(-\frac{1}{2\alpha} \sum_{S \in \mathcal{S}} 3\delta_S c_S \log n \right) \\ &\quad \cdot \prod_{S \in \mathcal{S}'} \exp \left(\frac{1}{2\alpha} 3\delta_S c_S \log n \right) \leq T. \end{aligned}$$

By linearity of expectation it now follows that $\mathbf{Exp}[\Phi_e] \leq \Phi_s$. Therefore, there exists a choice of sets from \mathcal{S}_j such that $\Phi_e \leq \Phi_s$. \square

Theorem 7. *Throughout the algorithm, the following properties hold.*

(i) *Every $j \in X$ of weight $w_j \geq 1$ is covered, that is, $j \in C$.*

(ii) $\sum_{S \in \mathcal{C}} c_S \leq (6 + o(1))\alpha \log m \log n$.

Proof. Initially, the value of the potential function Φ is at most $n \cdot n^{2/m} + n < n^2$, and hence it stays smaller than n^2 during the whole algorithm. Therefore, if $w_j \geq 1$ for some j during the process, then $j \in C$, since otherwise the contribution of the term n^{2w_j} itself would be at least n^2 . This proves part (i). To prove part (ii), note that by the same argument, throughout the algorithm

$$n \cdot \exp \left(\frac{1}{2\alpha} \sum_{S \in \mathcal{S}} c_S \chi_{\mathcal{C}}(S) - 3w_S c_S \log n \right) < n^2.$$

Therefore,

$$\sum_{S \in \mathcal{S}} c_S \chi_{\mathcal{C}}(S) \leq \sum_{S \in \mathcal{S}} 3w_S c_S \log n + 2\alpha \log n,$$

and the desired result follows from Lemma 5. \square

4 Derandomization

The choices of the various sets S to be added to \mathcal{C} after each iteration can be done deterministically and efficiently, by the method of conditional probabilities, c.f., e.g., [1], chapter 15. In fact, this can be done here in a very simple way. A close look at the proof in the last section shows that we can simply decide, after each weight augmentation, for each set whose weight has been increased in its turn, if we add it to \mathcal{C} or not, making sure that the potential function will not increase after each such choice. The details will appear in the full version of the paper.

We note that knowing in advance the set cover instance is crucial for making the online algorithm deterministic. To see this, consider the following example where the sets containing an element j are revealed only when the adversary gives element j to the algorithm. The set cover instance contains elements $\{1, 2, \dots, n\}$ and sets S_1, \dots, S_n . The adversary gives the algorithm the elements in the order $1, 2, \dots, n$. Assume inductively that the algorithm has used sets S_1, \dots, S_i for covering elements $1, \dots, i$. When element $i + 1$ is given, the sets containing it are S_{i+1}, \dots, S_n . Without loss of generality we can assume that the algorithm used set S_{i+1} for covering element $i + 1$. Clearly, the adversary can cover all the elements with a single set, S_n , yielding a competitive ratio of n .

5 Lower Bound

In this section we show that for every fixed $\delta > 0$ and every m and n satisfying

$$\log n \leq m \leq e^{n^{1/2-\delta}}, \quad (9)$$

there is a family \mathcal{F} of m subsets of X , $|X| = n$, so that the competitive ratio of any deterministic online algorithm for the (unweighted) online set cover problem with X and \mathcal{F} is at least

$$\Omega\left(\frac{\log n \log m}{\log \log m + \log \log n}\right). \quad (10)$$

Before describing the proof, we note that the assumption (9) is essentially optimal. Let OPT denote the value of the optimum (off-line) solution to the problem. Note, first, that the problem has a trivial algorithm with competitive ratio m (that simply takes all sets after the first element appears), showing that for $m < (\log n)^{1-\epsilon}$ the above lower bound (10) fails. (In fact, we may always assume that $m \geq \log_2 n$, since all the elements that lie in the same cell of the Venn diagram of the sets in \mathcal{F} can be replaced by a single element, without any change in the problem.) It is also easy to see that the problem has a simple algorithm with competitive ratio $O(\sqrt{n})$; when the first element arrives, we pick, repeatedly, all sets that cover at least \sqrt{n} members of X among those not covered so far. Note that after this process terminates, there are at most $OPT\sqrt{n}$ yet uncovered elements that can appear, and hence even if from now on we pick an arbitrarily chosen set for each new element, the algorithm will choose at most $\sqrt{n} + OPT\sqrt{n}$ sets altogether. (By being a bit more careful we can actually get an $O(\sqrt{n/OPT})$ -competitive algorithm this way. The details are left to the reader). This discussion shows that for $m > e^{n^{1/2+\delta}}$ the lower bound (10) also fails. Therefore, both inequalities in the assumption (9) are needed.

Proposition 8. *Let $X = \{0, 1, 2, \dots, n - 1\}$ be a set of $n = 2^k$ elements. For each $1 \leq i \leq k$, let F_i be the set of all elements j of X so that the i th bit in the binary representation of j is on, and*

let \mathcal{F} be the family of all k sets F_i . Then, the competitive ratio of the best deterministic algorithm for the online set cover problem (X, \mathcal{F}) is $|\mathcal{F}| = k = \log_2 n$.

Proof. The adversary starts by giving the number $n - 1$ in which all bits are on. If the algorithm covers it by F_{i_1} , then the adversary gives the number in which all bits are on besides the i_1 th bit. The algorithm covers it by F_{i_2} and the adversary gives the number in which all bits are on besides the i_1 th and i_2 th bits, and so on. Clearly, the algorithm will have to choose this way all k sets, while the optimal solution consists of only one set: the last set chosen by the algorithm. \square

The above proposition and some obvious modification of the family \mathcal{F} for bigger values of m implies that the lower bound (10) holds for all m satisfying, say, $\log_2 n \leq m \leq (\log_2 n)^3$. It thus remains to establish the lower bound for pairs n, m satisfying $(\log n)^3 \leq m \leq e^{n^{1/2-\delta}}$. This is done in what follows.

Let k, r be positive integers. Suppose $n \geq 2^k k r^2$, and let $X_1, X_2, \dots, X_{kr^2}$ be kr^2 pairwise disjoint blocks of elements in $X = \{1, 2, \dots, n\}$, each of size 2^k . For a block X_b and a bit location t , with $1 \leq t \leq k$, let $X_b(t)$ denote the set of all elements in X_b in which the t th bit is on. For each subset $R = \{b_1, b_2, \dots, b_r\}$ of size r of $\{1, 2, \dots, kr^2\}$, with $b_1 < b_2 < \dots < b_r$, and for each sequence of choices of bit locations $I = (i_1, i_2, \dots, i_r)$, where $1 \leq i_t \leq k$ for all t , define

$$F_{R,I} = \cup_{t=1}^r X_{b_t}(i_t).$$

Note that each such set contains elements from r blocks, and in each such block it contains half of the elements. Let \mathcal{F} denote the family of all sets $F_{R,I}$ as above. Therefore, $m = |\mathcal{F}| = \binom{kr^2}{r} k^r$.

We next show that given any deterministic algorithm, an adversary can choose kr elements in X , forcing the algorithm to pick kr sets from \mathcal{F} , while keeping the value of the optimum solution to be 1. The adversary starts by picking a block, say the first one, and by following the strategy described in the proof of Proposition 8 in this block. That is, the first chosen element is the member of X_1 in which all bits are on, when the algorithm covers it by a set in which the i_1 is on in X_1 , the adversary chooses the element of X_1 in which all bits are on besides i_1 , and so on. After k such steps the algorithm used already k sets. These sets contain elements in at most $1 + (r - 1)k < kr$ distinct blocks. The adversary will not choose any elements of these blocks from now on, pick another block, and repeat the same process of making k choices in this block. This can be repeated r times, while still enabling the adversary to cover all elements picked by one set, implying the desired result.

By adding, if needed, some extra $2^k k r^2$ elements and some of their subsets which we will not use, this implies the following.

Proposition 9. *For every positive integers k, r , and every n, m satisfying $n \geq 2^{k+1} k r^2$ and $2^{2^k k r^2} \geq m \geq \binom{kr^2}{r} k^r$, there is an example of an online set cover problem with n elements and m sets in which the competitive ratio of any deterministic algorithm is at least kr .*

Suppose, now, that n, m are large and satisfy (9). If $m \leq (\log n)^3$, the required lower bound (10) follows from Proposition 8, as mentioned after its proof. Otherwise, one can define $r = \Theta\left(\frac{\log m}{\log \log m + \log \log n}\right)$ and $k = \Omega(\log n)$ such that $n \geq 2^{k+1} k r^2$ and $2^{2^k k r^2} \geq m \geq \binom{kr^2}{r} k^r$. The required bound now follows from Proposition 9.

6 Concluding remarks

- We described a deterministic $O(\log m \log n)$ competitive algorithm for the online weighted set cover problem for a set X , $|X| = n$ and a family \mathcal{F} , $|\mathcal{F}| = m$, and showed that this is optimal, up to a $\log \log n + \log \log m$ factor. For some families of subsets \mathcal{F} , one can obtain online algorithms with better performance. It may be interesting to identify properties of the family \mathcal{F} that imply the existence of algorithms with better performance.
- In each weight augmentation step in the algorithm described in Section 3, the weight w_S of each set S is increased by a factor of $(1 + \frac{1}{nc_S})$. The factor n appearing here is simply for the sake of obtaining a better absolute constant in the analysis, and one can in fact augment the weight of S by a factor of $(1 + \frac{1}{c_S})$ without any real change in the performance. This is useful when we care about the efficiency of our algorithm, as it decreases the number of steps in which we have to add sets to the collection \mathcal{C} . In fact, it is possible to describe a slightly modified version of the algorithm where after the adversary presents an element j with $w_j < 1$, the weight of each set S containing j is increased from w_S to $w_S \cdot \exp(\frac{x}{c_S})$, where $x > 0$ is chosen so that after the augmentation $w_j = 1$. This, and the brief discussion in Section 4, enables the algorithm to consider all the sets containing j only once after the adversary presents it.
- The technique of converting an online fractional solution into a randomized algorithm (and later a deterministic one) used here can be applied when the fractional solution is monotone increasing during the algorithm. We believe that this method is likely to be useful in future applications as well.

Acknowledgements

The last author would like to thank Julia Chuzhoy, Eli Gafni, Sanjeev Khanna, Elias Koutsoupias, and Baruch Schieber for many stimulating discussions on the problem.

References

- [1] N. Alon and J. H. Spencer, **The probabilistic method**, Second Edition, Wiley, New York, 2000.
- [2] B. Awerbuch, Y. Azar, A. Fiat, and T. Leighton, Making commitments in the face of uncertainty: how to pick a winner almost every time, In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pp. 519-530, 1996.
- [3] A. Blum, On-line algorithms in machine learning, In: A. Fiat and G. Woeginger, editors, *Online algorithms - the state of the art*, Chapter 14, pp. 306–325, Springer, 1998.
- [4] A. Blum, Online learning tools for online algorithms, Dagstuhl Workshop on Online Algorithms, July 2002. (See <http://www-2.cs.cmu.edu/~avrim/surveys.html>.)

- [5] S. Chawla, A. Kalai, and A. Blum. Static optimality and dynamic search-optimality in lists and trees. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1-8, 2002.
- [6] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.
- [7] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, July 1998.
- [8] Y. Freund and R. E. Schapire. Game theory, on-line prediction and boosting. In *Proceedings of the 9th Annual Conference on Computational Learning Theory*, pp. 325-332, 1996.
- [9] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9:256-278, 1974.
- [10] L. Lovász. On the ratio of optimal and fractional covers. *Discrete Mathematics*, 13:383-390, 1975.
- [11] N. Littlestone and M. K. Warmuth. The Weighted Majority Algorithm. *Information and Computation*, 108:212–261, 1994.