On the Use of Randomization in the Online Set Cover Problem

By Simon Korman

Under the Supervision of Professor Uriel Feige
Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science

Thesis for the M.Sc. Degree
Submitted to the Scientific Council of the Weizmann Institute of Science
Rehovot, Israel 76100.

December 2004

Abstract

The (unweighted) *online set cover* problem was introduced by Alon *et.al.* in [AAA$^+$03]. It is best described as a game between an online player (algorithm) and an adversary. At the beginning of the game, both player and adversary are given a ground set of n elements $\mathcal{X} = \{1, 2, \ldots, n\}$ and a family of m subsets of $\mathcal{X}$, $F = \{s_1, s_2, \ldots, s_m\}$. At each round of the game, the adversary sends the player an element from $\mathcal{X}$. As the game proceeds, the player constructs a cover C_{ALG} according to the following rule. On the arrival of an element, if it isn't covered by the cover constructed so far, the online player must add to it (at least) one set that covers it. The player has no prior information regarding which of the n elements will be sent to it and in which order will they arrive.

The set $\mathcal{X}'$ of elements given by the adversary ($\mathcal{X}' \subseteq \mathcal{X}$), whose size is denoted by n', may be a strict subset of $\mathcal{X}$ ($n' < n$). At the end of the game, the adversary comes up with its own (optimal) cover of $\mathcal{X}'$, denoted by C_{OPT}. The goal of the algorithm is to construct a minimal sized cover and its performance is measured by the ratio $|C_{ALG}|/|C_{OPT}|$. The *competitive ratio* of the algorithm is the maximal such ratio, taken over all instances and all possible request sequences.

In [AAA$^+$03], Alon *et.al.* present a deterministic polynomial time algorithm with an accompanying lower bound, establishing that the *optimal deterministic competitive ratio* of any online algorithm is at most $O(\lg n \lg m)$ and at least $\Omega\left(\frac{\lg n \lg m}{\lg\lg n + \lg\lg m}\right)$.

In this thesis we explore the use of randomization in the online set cover problem, attempting to determine the *optimal randomized competitive ratio* ($ORCR$). It is clearly at most $O(\lg n \lg m)$ and by a simple lower bound we obtain - at least $\Omega(\lg m)$. Working towards closing this wide gap - on the upper end, we present 2 randomized algorithms, which imply $ORCR = O\Big(\lg n' \lg m\Big)$ and $ORCR = O\Big(k \lg \frac{m}{k}\Big)$, where k is the size of an optimal cover, improving on the previous upper bound when n' or k are small. On the lower end, the main contribution of this thesis is a hardness result, which implies that no randomized polynomial time algorithm has a competitive ratio which is $o(\lg n \lg m)$, unless $NP \subseteq BPP$.

Acknowledgements

I would first like to thank my advisor, Uri Feige, for introducing the field of online algorithms to me and being a remarkable guide throughout the year. Especially, for being very involved in my work and being available at any time for many inspiring discussions. Also, for making possible my visit to him at Microsoft, Redmond. I would like to thank Moni Naor for being my advisor for the last few months while Uri was away on a sabbatical.

My best friends at the computer science and mathematics department - Daniel, Ilan, Erez, Yoram, Alex, Uri and Yaron together with many other fellow students, made my studies here at the Weizmann Institute a great pleasure.

Finally, I would like to thank my family and especially - my wife Sivan, my parents Betty and Stanley and my newborn son Amit, for their endless love.

This work was supported in part by a grant from the German-Israeli Foundation for Scientific Research and Development.

Contents

Chapter 1

Introduction

1.1 The Online Set Cover Problem

Given a ground set of elements $\mathcal{X} = \{1, 2, \ldots, n\}$ and a family of subsets of $\mathcal{X}$, $F = \{s_1, s_2, \ldots, s_m\}$, *set cover* is the problem of finding a minimal number of subsets out of $\mathcal{F}$, whose union contains $\mathcal{X}$. Set cover is a classic NP-hard problem that was studied extensively in the literature. Polynomial time algorithms that give an approximation ratio of $\ln n$ were given by Johnson, Chvátal and Lovász in [Joh74, Chv79, Lov75]. Tighter analyses on these algorithms were given by Srinivasan and Slavik [Sri, Sla96] and hardness of approximation results were given by Lund and Yannakakis, Raz and Safra and Feige in [LY93, RS97, Fei98].

For any subset $\mathcal{X}'$ of $\mathcal{X}$, a *cover* of $\mathcal{X}'$ is a collection of sets from $\mathcal{F}$, whose union contains $\mathcal{X}'$. In the weighted version of set cover, the *cost* of a cover is defined to be the sum of the costs of the sets it contains (in the unweighted version, it is simply the size of the cover).

The *online set cover* problem was introduced by Alon *et.al.* in [AAA$^+$03] to be the following game between an online player (algorithm) and an adversary. Given an (offline) set cover instance $I = \langle \mathcal{X}, F \rangle$, the adversary gives the algorithm elements from $\mathcal{X}$, one by one. Once the algorithm receives a new element, it has to immediately cover it by a set from $\mathcal{F}$ containing it. If we denote by $\mathcal{X}' \subseteq \mathcal{X}$ the set (sequence) of elements given by the adversary throughout the game, the algorithm's objective is to construct a cover of $\mathcal{X}'$, whose cost is minimal. Note that in general, $\mathcal{X}'$ may be a strict subset of $\mathcal{X}$, since otherwise the algorithm could use standard offline techniques, e.g. the greedy algorithm (if it is restricted to polynomial time) or even the brute force naive algorithm that fully solves the problem (if it is allowed exponential time).

We list here some basic notations that will be used throughout this thesis.

$\mathcal{I}^{on}$	An online set cover instance
$\mathcal{I}^{off}$	An offline set cover instance
$\mathcal{X}$	The ground set of elements of a set cover instance
$\mathcal{X}'$	The set (sequence) of elements ($\mathcal{X}' \subset \mathcal{X}$) that are sent by the online adversary
$\mathcal{F}$	The family of sets of a set cover instance
n	The size of the ground set $\mathcal{X}$
n'	The size of the partial set $\mathcal{X}'$
m	The size of the family $\mathcal{F}$
k	The optimal cover size of $\mathcal{X}'$ ($\mathcal{X}$)
$[r]$	The set $\{1, 2, \ldots r\}$
$\mathcal{H}_l$	The lth harmonic number $1/1 + 1/2 + 1/3 + \ldots + 1/l$

As with other online problems, one can define two different models, that relate to the extent to which the input is hidden from the online player.

The known-instance model The online algorithm knows in advance the set cover instance $I = \langle \mathcal{X}, S \rangle$, however, it has no in-advance knowledge regarding the sequence of elements $\mathcal{X}'$ given by the adversary. Namely, when a new element arrives, the algorithm can uniquely identify it with a ground set element.

The unknown-instance model The online algorithm has *no* prior knowledge regarding the set cover instance or its structure, in addition to having no in-advance knowledge regarding the sequence of elements $\mathcal{X}'$ given by the adversary. Namely, when a new element arrives, it is accompanied with the information of which sets cover it.

It is easy to verify on one hand, that any algorithm for the unknown-instance model can be used for the known-instance model (meaning that upper bounds for the unknown case hold for the know case) and on the other hand, that any lower bound obtained for the known-instance model holds for the unknown-instance model. For this reason, we prove our upper bounds in the unknown-instance model and lower bounds - in the known-instance model.

The online set cover problem captures many practical scenarios. The following example is from [AAA$^+$03]. Consider severs in a network that provide a service. There is a set of potential clients that may need the service and each server can give the service (at once) to a subset of the users. Upon the arrival of a client, the network manager has to decide which server to activate (incurring an activation cost) so that the client receives the service it requested. The network manager might know in advance the list of potential clients and the set of servers (known/unknown instance model), however, he doesn't know in advance which clients will indeed request the service.

We give some background and previous results on the online set cover problem in section 1.3.

1.2 Randomization in Online Computation

An *online* algorithm is an algorithm which works under the following handicap: the algorithm is not allowed to immediately examine the entire problem which it is trying to solve. The problem is revealed to it incrementally and the algorithm's solution must also be produced incrementally. This framework is frequently used to model problems which evolve over time. The online algorithm operates under the handicap of an uncertain future. Since decisions about the output are made with incomplete knowledge about the entire input, an online algorithm often can not produce an optimal solution. Such an algorithm can only hope to approximate the performance of an optimal algorithm that sees all the inputs in advance. We denote an algorithm of the second type, namely, one that operates with complete knowledge of the input an *offline* algorithm.

Two well known examples are: the *Paging* problem, where an operating system has to decide which pages to keep in a cache during the execution of a program so as to maximize cache utilization [ST85, FKL$^+$91, MS91] and the *Load Balancing* problem, in which it has to be decided how to schedule a job on some set of machines, when future jobs are unknown so as to minimize the last completion time [BFKV95, KPT96, Alb97]. Some other classical problems are: *List Accessing*, *k-Server*, *Call Admission*, *Circuit Routing* and *Portfolio Selection*. In addition, online versions had been suggested for many standard combinatorial optimization problems. Examples are *Graph Coloring* [LST89, HS92, BFL96], *Bipartite Matching* [KVV90] and *Bin Packing* (surveyed in [CW98]). The online set cover problem that we discuss in this thesis is another such example.

1.2.1 Competitive Analysis

Competitive analysis is a type of worst case analysis where the performance of an online algorithm is compared to that of the optimal offline algorithm. This approach to analyzing online problems was initiated by Sleator and Tarjan in [ST85].

The measure of performance used in competitive analysis is the *competitive ratio*. For a given cost minimization problem $\mathcal{P}$ and an online algorithm $\mathcal{A}$ for $\mathcal{P}$, let $C_{\mathcal{A}}(\rho)$ be the cost incurred by the algorithm $\mathcal{A}$ on an instance ρ of the problem $\mathcal{P}$. Similarly, let $C_{OPT}(\rho)$ be the cost incurred by an optimal offline algorithm for $\mathcal{P}$ on the instance ρ.

Definition 1.2.1. A deterministic online algorithm $\mathcal{A}$ for a problem $\mathcal{P}$ is said to be *c-competitive* if there exists a constant d such that $C_{\mathcal{A}}(\rho) \leq c \cdot C_{OPT}(\rho) + d$ for every instance ρ of $\mathcal{P}$. [1]

The goal in competitive analysis is to find, for a particular problem, an algorithm with lowest possible competitive ratio. Since the competitive ratio is a worst case measure, for the purposes of analysis, we assume that the input sequence is generated by a malicious adversary, who forces the algorithm to perform as badly as possible.

1.2.2 Randomized Competitive Analysis

A randomized algorithm is one which bases its decision making on the outcome of random coin flips. Randomization enables an algorithm to perform well on average, where the average is taken over the algorithm's random coin flips. It has proven itself to be a useful tool in the design of algorithms, both online and offline. See [MR97] for an introduction to randomized algorithms.

Randomization is useful since it allows the online algorithm to hide information about its state from the adversary. For a randomized algorithm $\mathcal{A}$, the cost it pays when running on input ρ, $C_{\mathcal{A}}(\rho)$ is a random variable. The following definitions follow.

Definition 1.2.2. A randomized online algorithm $\mathcal{A}$ for a problem $\mathcal{P}$ is said to be *c-competitive* if there exists a constant d such that $\mathrm{E}\big[C_{\mathcal{A}}(\rho)\big] \leq c \cdot C_{OPT}(\rho) + d$ for every instance ρ of $\mathcal{P}$.

The quality of an online algorithm, whether deterministic or randomized, is measured by its *competitive ratio*, defined as follows.

Definition 1.2.3. The *competitive ratio* of an algorithm $\mathcal{A}$, denoted by $CR_{\mathcal{A}}$, is defined to be: $\inf\{\ c \mid \mathcal{A}$ is *c-competitive*$\}$.

The question of how much power the adversary has in terms of information about the online algorithm's random choices introduces some subtlety to definitions (1.2.2) and (1.2.3). In their seminal paper [BDBK$^+$94], Ben-David *et.al.* introduce three kinds of adversarial models that play against a randomized algorithm and prove theorems that relate their power. On one end is the *oblivious* adversary. This is the kind of adversary, that has the least knowledge about the online player (algorithm) A. Based on the knowledge of the algorithm

[1]The additive constant d is necessary in the standard definition of competitive analysis, but could be removed (always taken to be zero) in the context of this thesis.

A - in particular, the probability distributions that A uses - such an adversary must choose a finite request sequence ρ in advance. The remaining two kinds of adversaries are *adaptive* adversaries, where the input sequence ρ need not be predetermined, namely, the current request that the adversary sends can depend on the online algorithm's answers to previous requests. We focus in this thesis on the oblivious adversarial model.

For several problems, randomization has yielded online algorithms which outperform their best possible deterministic counterparts. One such case is the abstract model for competitive analysis of online computation, called metrical task systems (MTS), formulated by Borodin, Linial and Saks [BLS92]. The authors introduce a number of general concepts and techniques and show how randomization helps in the case of uniform MTSs.

1.2.3 A Technique for Obtaining Lower Bounds

Competitive Analysis and Zero Sum Games

Competitive analysis of an online problem can be viewed as the analysis of a two-person zero-sum game. In particular, determining the optimal competitive ratio of an online problem is strongly related to determining the *value* of the corresponding game.

Informally, a game is called a *zero-sum* game if in any outcome of the game, the sum of payoffs for all players equals zero. Specifically, in a *two-person zero-sum* game, the payoff for one player is exactly the (additive) inverse of the payoff for the other player and this is the reason that such games are suitable for representing a conflict between an online player and an adversary. In this section, we show how online computation can be modelled in terms of zero-sum games and then, how Yao's principle for obtaining lower bounds on randomized online algorithms can be derived, using standard game theoretic tools.

Consider an online cost minimization problem. We view it as a two-person zero-sum game in the following manner. The first player, who is the online player (algorithm) seeks a strategy that minimizes its cost. The payoff for the first player (who we assume is a payoff maximizer) can be defined in several ways, the conventional one being the reciprocal of its competitive ratio. The second player, who is the adversary, seeks a strategy that minimizes the online player's payoff. Clearly, the players are in strict opposition and the game is zero-sum.

We consider a finite game in strategic form, in which the payoffs for the first player are specified by the matrix $G = (h_{ij})$, where the row index, i, enumerates the algorithm's pure strategies, while the column index, j, enumerates the adversary's pure strategies.

Definition 1.2.4. A pair (i^*, j^*) of pure strategies is called a *saddle point* if for all i and j

$$h(i, j^*) \leq h(i^*, j^*) \leq h(i^*, j)$$

$$\Big(\text{equivalently, if:} \quad \max_i \min_j h(i,j) = h(i^*, j^*) = \min_j \max_i h(i,j)\Big).$$

When a saddle point exists, we say that the game has a "*value* with pure strategies" (where the value is $h(i^*, j^*)$) or we say that the game has a "pure strategy *equilibrium*". In general, saddle points do not exist. In the case a saddle point exists, it need not be unique, however - the corresponding values are identical, by definition.

A *mixed strategy*, is simply a distribution over the possible pure strategies for a player.

Definition 1.2.5. Let $G = (h_{ij})$ be a two-person zero-sum game. For mixed strategies $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_n)$, for players 1 and 2, respectively, we define

$$H(x, y) = \sum_i \sum_j h(i, j) x_i y_j$$

to be the *expected payoff* for player 1, under x and y.

The resulting two-person zero-sum game with players 1 and 2 using $H(x, y)$ as a payoff function is called the *mixed extension of G*.

Definition 1.2.6. A pair (x^*, y^*) of mixed strategies for the mixed extension of G is called a *saddle point* if for all x and y

$$H(x, y^*) \leq H(x^*, y^*) \leq H(x^*, y)$$

$$\Big(\text{equivalently, if:} \quad \max_x \min_y H(x,y) = H(x^*, y^*) = \min_y \max_x H(x,y)\Big).$$

When such a saddle point (mixed strategy equilibrium) exists, we say that the mixed extension of G has a *value*, which is $H(x^*, y^*)$. The celebrated minimax theorem of Morgenstern and von-Neumann [MvN44] states that such a value always exists.

Theorem 1.2.7. *(Minimax Theorem [MvN44]) Every finite two-person zero-sum game has a value. That is:*

$$\max_x \min_y H(x, y) = \min_y \max_x H(x, y).$$

Notice that theorem 1.2.7 is only existential. Nevertheless, there are various algorithms for computing optimal mixed strategies, some of which are linear programming based. A simple observation from the minimax theorem is that if player 1 (respectively player 2) knows that player 2 (respectively player 1) uses (an optimal) mixed strategy, one of his optimal (mixed) strategies is pure. This is established in the following corollary.

Corollary 1.2.8. *(Loomis' Lemma [Loo46]) Let x^* and y^* be optimal strategies for players 1 and 2. Then:*

$$\max_x \min_j H(x,j) = H(x^*,y^*) = \min_y \max_i H(i,y) \tag{1.1}$$

Another conclusion of the minimax theorem is the following.

Corollary 1.2.9. *(Yao's Inequality [Yao77]) For any mixed strategy y for player 2*

$$\max_x \min_j H(x,j) \leq \max_i H(i,y) \;, \tag{1.2}$$

and in the case that player 1 is a cost minimizer:

$$\min_x \max_j H(x,j) \geq \min_i H(i,y) \;. \tag{1.3}$$

Yao's Principle

The application of Yao's inequality as a tool for obtaining lower bounds is straightforward in standard complexity. However, for the purpose of competitive analysis, it is much more delicate.

Theorem 1.2.10. *(Yao's principle for online cost minimization problems) Let P be any online cost minimization problem that can be formulated as a finite two-person zero-sum game. Let ALG be any randomized algorithm for P and let $y(j)$ be any probability distribution over request sequences. Then, the following lower bounds hold:*

$$CR_{ALG} \;\geq\; \min_i E_{y(j)}\left[\frac{C_{ALG_i}(\sigma_j)}{C_{OPT}(\sigma_j)}\right] \tag{1.4}$$

$$CR_{ALG} \;\geq\; \min_i \frac{E_{y(j)}\left[C_{ALG_i}(\sigma_j)\right]}{E_{y(j)}\left[C_{OPT}(\sigma_j)\right]} \tag{1.5}$$

Proof. In order to prove inequality (1.4), we first formulate P as a two-person zero-sum game as we did previously, with the payoff function $h(i,j) = \frac{C_{ALG_i}(\sigma_j)}{C_{OPT}(\sigma_j)}$, in which player 1 is clearly a *payoff minimizer.* Now, we have:

$$CR_{ALG} = \min_{x(i)} \max_j \frac{\mathrm{E}_{x(i)}\left[C_{ALG_i}(\sigma_j)\right]}{C_{OPT}(\sigma_j)} = \min_{x(i)} \max_j \mathrm{E}_{x(i)}\left[\frac{C_{ALG_i}(\sigma_j)}{C_{OPT}(\sigma_j)}\right] \geq \min_i \mathrm{E}_{y(j)}\left[\frac{C_{ALG_i}(\sigma_j)}{C_{OPT}(\sigma_j)}\right]$$

where the first equality is by the definition of the competitive ratio of a randomized algorithm and the last inequality follows by an application of Yao's inequality (1.3).

In order to prove inequality (1.5), we again formulate P as a two-person zero-sum game, for every positive constant c, with a the payoff function $h_c(i,j) = c \cdot C_{OPT}(\sigma_j) - C_{ALG_i}(\sigma_j)$, which clearly makes player 1 a *payoff maximizer.* For each c and each resulting game, denote by $value(c)$ the value of the game that is defined with respect to the payoff function h_c. According to Loomis' lemma (1.1), we have that:

$$value(c) = \max_{x(i)} \min_j H_c(x,j) = \max_{x(i)} \min_j \mathrm{E}_{x(i)}\big[h_c(i,j)\big]. \tag{1.6}$$

A first observation that follows from (1.6) and from the definition of h_c is that

$$CR_{ALG} \leq c \qquad \Longleftrightarrow \qquad value(c) \geq 0 \tag{1.7}$$

One the other hand, we prove that

$$\text{For every } c \text{ such that } \ c < \min_i \frac{\mathrm{E}_{y(j)}\big[C_{ALG_i}(\sigma_j)\big]}{\mathrm{E}_{y(j)}\big[C_{OPT}(\sigma_j)\big]} \ : \qquad value(c) < 0 \ . \tag{1.8}$$

The combination of observations (1.7) and (1.8) immediately implies that inequality (1.5) holds and the proof is completed.

We are left with proving observation (1.8). This is done as follows:

$$value(c) = \max_{x(i)} \min_j \mathrm{E}_{x(i)}\big[h_c(i,j)\big] \leq \max_i \mathrm{E}_{y(j)}\big[h_c(i,j)\big] =$$
$$\max_i \mathrm{E}_{y(j)}\big[c \cdot C_{OPT}(\sigma_j) - C_{ALG_i}(\sigma_j)\big] < 0 \ , \tag{1.9}$$

where the first inequality is by an application of Yao's inequality (1.2) and the second inequality is simply a rearrangement of the condition $c < \min_i \frac{\mathrm{E}_{y(j)}[C_{ALG_i}(\sigma_j)]}{\mathrm{E}_{y(j)}[C_{OPT}(\sigma_j)]}$.

■

1.3 Previous Work on the OSC Problem

The paper [AAA+03] of Alon *et.al.* was the one that introduced the online set cover problem. An earlier paper [AAFL96] by Awerbuch *et.al.* discusses a different version of the online set cover problem (that might better be named *online max k-cover*) which deals with maximum benefit rather than with minimum cost. Namely, in this version, the goal of the algorithm is to pick k sets so as to maximize the number of covered elements, where the algorithm gets a single credit for each element, that is covered by a set that was picked during or before the element's arrival. This thesis deals only with the minimum cost version of [AAA+03].

1.3.1 A Competitive Deterministic Algorithm

The main result in [AAA+03] is an $O(\lg n \lg m)$-competitive deterministic algorithm for the online set cover problem, which is actually $O(\lg n \lg d)$-competitive, if each element appears in at most d sets.

The idea of the algorithm of Alon *et.al.*, is as follows. It starts by producing an online fractional solution to the problem, which yields a randomized algorithm by a technique that is a variation on the conventional technique of randomized rounding. Finally, the randomized algorithm is derandomized. We note that the algorithm runs in polynomial time.

Each set s will be assigned a weight ω_s and similarly, each element j will have a weight ω_j, which is defined to be, at any stage, the sum of the weights of the sets containing that element. Denote by $\mathcal{S}_j$ the collection of sets that contain the element j and by $\mathcal{C}$ the collection of sets which forms the constructed cover.

Algorithm *AAABN*

◇ Initialization: $\mathcal{C} = \phi$ and for each $s \in \mathcal{S}$: $\omega_s = \frac{1}{2m}$.

◇ Define the potential function, $\Phi = \sum_{j \notin \mathcal{C}} n^{2w_j}$.

◇ On the arrival of an element j:

1. If $w_j \geq 1$, do nothing and wait for the arrival of a new element.
2. Else, ($w_j < 1$), perform a *weight augmentation*:
   (a) Let k be the minimal integer for which $2^k \cdot w_j > 1$.
   (b) For each set $s \in \mathcal{S}_j$, $w_s = 2^k \cdot w_s$.
   (c) Choose from $\mathcal{S}_j$ at most $4 \lg n$ sets to $\mathcal{C}$, so that the value of the potential function Φ does not exceed its value before the weight augmentation.

◇ Return the cover $\mathcal{C}$.

The authors showed that the algorithm produces a legal cover and proved its competitive ratio. Their main effort was in showing how the sets in step (c) are chosen. They first give a probabilistic proof for the existence of $4 \lg n$ such sets. Then, they claim that choosing such sets deterministically is possible, using the method of conditional probabilities (which is described in [AS00], chapter 15).

1.3.2 A Lower Bound on Deterministic Algorithms

The above deterministic algorithm is accompanied in [AAA+03] by a nearly matching $\Omega\left(\frac{\lg n \lg m}{\lg \lg n + \lg \lg m}\right)$ lower bound which holds for any m and n that satisfy $\lg n \leq m \leq e^{n^{0.5-\delta}}$ for any fixed $\delta > 0$ (these include most interesting values of m and n).

Description

The lower bound is constructed as follows. Let k and r be parameters whose value we determine later. Consider an instance, whose ground set of elements $\mathcal{X}$ consists of kr^2 distinct copies of all the binary vectors of length k. We now define the family of sets $\mathcal{F}$. We say that a set *hits* copy j ($1 \leq j \leq kr^2$) at index l ($1 \leq l \leq k$) if it covers out of the vectors of the jth copy, exactly those, whose lth coordinate is 1. Each set will hit exactly r out of the kr^2 sets, each at a single index. Formally, for each pair of r-tuples $(c_1, \ldots, c_r)$ and $(ind_1, \ldots, ind_r)$ (where the c_i correspond to distinct copy numbers and the ind_i correspond to (non distinct) indices), there exists a set that hits copy c_i at index ind_i for each $1 \leq i \leq r$.

Next, they show that given any deterministic algorithm DET, an adversary can choose a sequence of kr elements, which force DET to pick kr sets (every element that arrives is uncovered), while keeping the value of the optimal solution to be 1. The adversary works as follows. It sends to DET elements (vectors) in r rounds, k vectors that belong to a single copy during each round. Starting with round 1, the adversary picks an arbitrary copy and sends the vector that is all ones from that copy. DET is forced to pick some set that hits this copy, say at index i_1. Next, the adversary sends the vector that is all ones, except for the coordinate i_1. Again, DET is forced to pick some set that hits this copy, say at index i_2. Next, the adversary sends the vector that is all ones, except for the coordinates i_1, i_2. The adversary continues in this manner and the round ends, when the adversary has sent k vectors, the last one being a vector that is all zero, except for a single coordinate. During the round, DET picked k sets, each hitting r copies. Altogether, DET could have hit at most $1 + (r-1)k < kr$ distinct copies. In the next stage, the adversary picks an arbitrary copy, that wasn't hit by DET. Since there are kr^2 copies altogether, this process can continue for r rounds. Notice that DET is forced to pick kr sets, while the adversary can cover all elements by a single set - the set that hits each of the r chosen copies at the last surviving coordinates.

By carefully setting $r = \Theta\left(\frac{\lg m}{\lg \lg n + \lg \lg m}\right)$ and $k = \Omega(\lg n)$ they obtain the desired bound for large enough m and n. See [AAA+03] for further details.

1.4 Our Results

Typically, in the field of online computation, we seek three types of results. The first is a proof that no algorithm can achieve a competitive ratio lower than a given value - This is a *lower bound* proof. This kind of result is achieved by defining an instance together with an adversary that forces any algorithm to incur the stated competitive ratio on this instance. The second is a rigorous proof that an algorithm for a particular problem achieves a certain competitive ratio - This is an *upper bound* proof. When we have found matching upper and lower bounds - we have reached our goal. In many cases, there still exists a gap between the known bounds and then we seek a third type of result. That is, an *algorithm lower bound*, showing that the competitive ratio of a certain algorithm isn't lower than a given value.

Chapter 2 contains the results of the first type, while Chapter 3 contains the results of the second and third types.

The goal of this thesis was to explore the randomized approach to the online set cover Problem. As noted in section 1.3, all previous work on this problem was done under the deterministic model. We have found several advantages and several limits to the randomized approach and we summarize them in tables 1.1 and 1.2.

Specifically, on the negative side we prove a general $\Omega(\lg m)$ lower bound on the *optimal randomized competitive ratio* ($ORCR$) and we prove a hardness result, stating that unless $NP \subseteq BPP$, the competitive ratio of any randomized algorithm is at least $\Omega(\lg n \lg m)$. These negative results apply to the known-instance model.

On the positive side, we designed the randomized algorithms *RandSC* and *Uniform*, the first being polynomial time and the second - exponential time ($\Theta(m^k)$ per element), provably achieving competitive ratios of $O(\lg n' \lg m)$ and $O(k \lg \frac{m}{k})$, correspondingly, against oblivious adversaries. We prove that these ratios cannot be achieved by deterministic algorithms (implicit in [AAA+03]). We also prove the analyses of these algorithms to be tight. In addition, both algorithms work in the unknown-instance model.

As can be seen from these tables, for randomized algorithms there still exists a gap between the lower $\lg m$ bound and the upper $k \lg \frac{m}{k}$ and $\lg n' \lg m$ bounds. However, our hardness result for polynomial time algorithms implies that the only way one can hope to beat the $\lg n' \lg m$ upper bound, is via a randomized algorithm, whose running time is super-polynomial. The author is unaware of any online problems, for which the combination of randomization and super-polynomial time has yielded better competitive ratios than one could obtain, using only one of these ingredients. This might indicate that the above upper bounds are tight, with respect to any randomized online algorithm, implying that $ORCR = \Theta\big(\lg n \lg m\big)$ and resolving the question to what extent does randomization help in the online set cover problem.

	Randomized	**Deterministic**
1	Algorithm *RandSC* is $O(\lg n' \lg m)$-competitive against an oblivious adversary.	A competitive ratio of $O(\lg n' \lg m)$ is impossible for any deterministic algorithm.
2	Algorithm *Uniform* is $O(k \lg \frac{m}{k})$-competitive against an oblivious adversary.	A competitive ratio of $O(k \lg \frac{m}{k})$ is impossible for any deterministic algorithm.
3	Algorithm *RandSC* (*Uniform*) is $O(\lg n' \lg m)$-competitive $\big(O(k \lg \frac{m}{k})$-competitive$\big)$ even under the unknown-instance model.	In the unknown-instance model, the competitive ratio of any deterministic algorithm is at least m.

Table 1.1: **Advantages of randomization in the online set cover problem**

- The competitive ratio of any randomized algorithm is at least $\Omega(\lg m)$.
- Unless $NP \subset BPP$, the competitive ratio of any randomized polynomial time algorithm is at least $\Omega(\lg n \lg m)$.

Table 1.2: **Limits of randomization in the online set cover problem**

Chapter 2

Lower Bounds and Hardness Results

2.1 Lower Bounds on Deterministic Algorithms

The most general bound is the one given in [AAA+03] and which we proved in subsection 1.3.2. We state it here as a theorem.

Theorem 2.1.1. *For any $\delta > 0$ and any m and n that satisfy $\lg n \leq m \leq e^{n^{0.5-\delta}}$, the competitive ratio of any deterministic algorithm for online set cover is at least $\Omega\left(\frac{\lg n \lg m}{\lg\lg n + \lg\lg m}\right)$.*

We use the same construction that yielded the above lower bound in order to show that the competitive ratios that are achieved by the randomized algorithms that we present in chapter 3, cannot be achieved without the use of randomness.

Theorem 2.1.2.

1. *A competitive ratio of $O(\lg n' \lg m)$ is impossible for any deterministic algorithm.*
2. *A competitive ratio of $O(k \lg \frac{m}{k})$ is impossible for any deterministic algorithm.*

Proof. Using the instance from subsection 1.3.2, we show that any deterministic algorithm cannot attain a competitive ratio as stated in the theorem. We remind that in the instance $k = C_{OPT} = 1$, $n' = kr = \Omega\left(\frac{\lg n \lg m}{\lg\lg n + \lg\lg m}\right)$ and therefore, it will suffice to show that the number of chosen sets is $\omega(\lg n' \lg m) = \omega\left(\lg\left(\frac{\lg n \lg m}{\lg\lg n + \lg\lg m}\right) \cdot \lg m\right)$. However, we saw that any deterministic algorithm is forced to pick $n' = kr = \Omega\left(\frac{\lg n \lg m}{\lg\lg n + \lg\lg m}\right)$ sets. ∎

We use another example from [AAA+03] that shows that deterministic algorithms are of no use in the unknown-instance model. Namely, any such algorithm might as well naively choose all the sets at once on the arrival of the first element.

Theorem 2.1.3. *In the unknown-instance model, the competitive ratio of any deterministic algorithm is at least m.*

Proof. Consider an instance where the number of sets equals the number of elements. In the unknown-instance model, the algorithm is given an element and is told to which of the sets it belongs only on its arrival. In this instance, each arriving element is said to be contained exactly in each of the sets that the algorithm has not yet chosen. If an adversary acts in this manner and sends all of the m elements, clearly, the algorithm will pick all m sets, while an offline solution can contain only the last set chosen by the algorithm.

■

2.2 A $\lg m$ Lower Bound on Randomized Algorithms

2.2.1 Example Definition and Terminology

We look at the Set-Cover instance $\mathcal{I}_k = \langle \mathcal{X}_k, \mathcal{F}_k \rangle$ where: $\mathcal{X}_k$ is the set of all non-empty binary strings of length at most k. $\mathcal{F}_k$ is a collection of 2^k sets of elements, each indexed by a different binary string of length k and each such set, contains the k elements of $\mathcal{X}_k$, which are prefixes of the set's index.

If we denote by n and m the number of elements and sets correspondingly, we have that: $n = |\mathcal{X}_k| = 2 \cdot 2^k - 2$ and $m = |\mathcal{F}_k| = 2^k$.

2.2.2 Proof of Lower Bound

We prove a lower bound using Yao's Principle for online algorithms, which we proved in subsection 1.2.3. We specify a distribution ρ_k of inputs and analyze the competitiveness of some optimal deterministic algorithm with respect to ρ_k. The distribution of inputs ρ_k will be generated as follows:

- Pick a single set s from $\mathcal{F}_k$ uniformly at random
- Send, one by one, all of the k prefixes of the index of s, in order of increasing length

Regarding an optimal deterministic algorithm, we will assume that it is lazy. Namely that it doesn't choose unnecessary sets (i.e. if the arriving element has already been covered, it doesn't pick any set at all, and otherwise - it picks a single set to cover the element). We prove in claim A.0.4 that any deterministic algorithm can be assumed to be lazy, namely, that it could, if needed, be slightly altered, forcing it to have this property, while not increasing the overall number of sets it picks.

Note that for any input generated according to ρ_k, an optimal *off-line* algorithm can cover all given elements using a *single* set by simply choosing at the first round the set whose index is the last given element.

On the other hand, we show that our optimal deterministic *online* algorithm, will need to pick an expected number of $(k+1)/2$ sets for the cover. We fix our analysis to an optimal deterministic online algorithm, with the above property, name it DET.

Denote by $e_1, e_2, \ldots, e_k$ the k elements that arrive. If we denote by A_i the event that DET chose a set on the arrival of e_i (i.e. the element is uncovered on its arrival), by the linearity of expectation, the following holds.

$$\mathrm{E}[C_{DET}] = \sum_{i=1}^{k} \Pr[A_i] \tag{2.1}$$

Trivially, $\Pr[A_1] = 1$. For any $2 \leq i \leq k$: denote by i' the index of the last element $e_{i'}$ ($i' < i$) which was uncovered on its arrival. Let s' be the set chosen on the arrival of $e_{i'}$. Whether the event A_i happens is fully determined by whether s' contains the element e_i. The probability of this happening is exactly the fraction of sets that contain e_i out of the sets that contain e_{i-1}, which is, in our case: $1/2$.

Plugging this into (2.1), we get:

$$\mathrm{E}[C_{DET}] = \sum_{i=1}^{k} \Pr[A_i] = 1 + \sum_{i=2}^{k} \frac{1}{2} = \frac{k+1}{2} \tag{2.2}$$

This gives us the following desired theorem:

Theorem 2.2.1. *The competitive ratio of any randomized algorithm against an oblivious adversary for the online set cover problem is at least* $\frac{\lg n}{2}$ $\left(\frac{1+\lg m}{2}\right)$.

2.3 Hardness Results for Polynomial Time Algorithms

In this section, we give a hardness result of $\Omega(\lg n \lg m)$ for the competitive ratio of any randomized *polynomial* time algorithm for the online set cover problem. How do we interpret this result?

Regarding deterministic algorithms, Alon *et.al.* in [AAA+03] give a lower bound of $\Omega\left(\frac{\lg n \lg m}{\lg\lg n + \lg\lg m}\right)$ on the competitive ratio of any deterministic algorithm (regardless of its running time), which holds for most interesting values of m and n. Our hardness result strengthens their result, under the $P \neq NP$ assumption.

When we look at the general result, meaning, the hardness for *randomized polynomial* time algorithms under the $NP \nsubseteq BPP$ assumption, its consequence is the following. If indeed it is the case that the use of randomization in the online set cover problem gives a significant advantage over the deterministic approach, we may narrow our search for such a randomized algorithm to randomized algorithms that use super polynomial time.

We note here that this hardness result of $\Omega(\lg n \lg m)$ doesn't contradict the upper bound of $O(\lg n' \lg m)$ that we achieve in chapter 3.3, since in the instances we build here, $\lg n$ and $\lg n'$ are of the same order of magnitude.

The main ingredient we use is the following reduction.

2.3.1 A Reduction from Set Cover to Online Set Cover

Our reduction begins from a Set Cover instance $I^{off} = \langle X^{off}, F^{off} \rangle$ consisting of N elements and M sets and whose optimal cover size is k. We denote

$$\mathcal{X}^{off} = \{e_1, e_2, \ldots, e_N\} \qquad F^{off} = \{s_1, s_2, \ldots, s_M\}$$

Define $N-1$ distinct copies of I^{off} : $I^{(1)off}, I^{(2)off}, \ldots, I^{(N-1)off}$ in the following manner.

$$I^{(i)off} = \langle \mathcal{X}^{(i)off}, F^{(i)off} \rangle \qquad \text{such that:}$$

$$\mathcal{X}^{(i)off} = \{e^i_1, e^i_2, \ldots, e^i_N\} \qquad F^{(i)off} = \{s^i_1, s^i_2, \ldots, s^i_M\} \qquad (\text{where } e^i_j \in s^i_l \iff e_j \in s_l)$$

We use the convention that upper indices correspond to the offline instance number, whereas lower indices correspond to the number of a set or element within an instance. We next describe the online instance $I^{on} = \langle \mathcal{X}^{on}, F^{on} \rangle$. We begin with an informal description (The formal one appears in Figure 2.2).

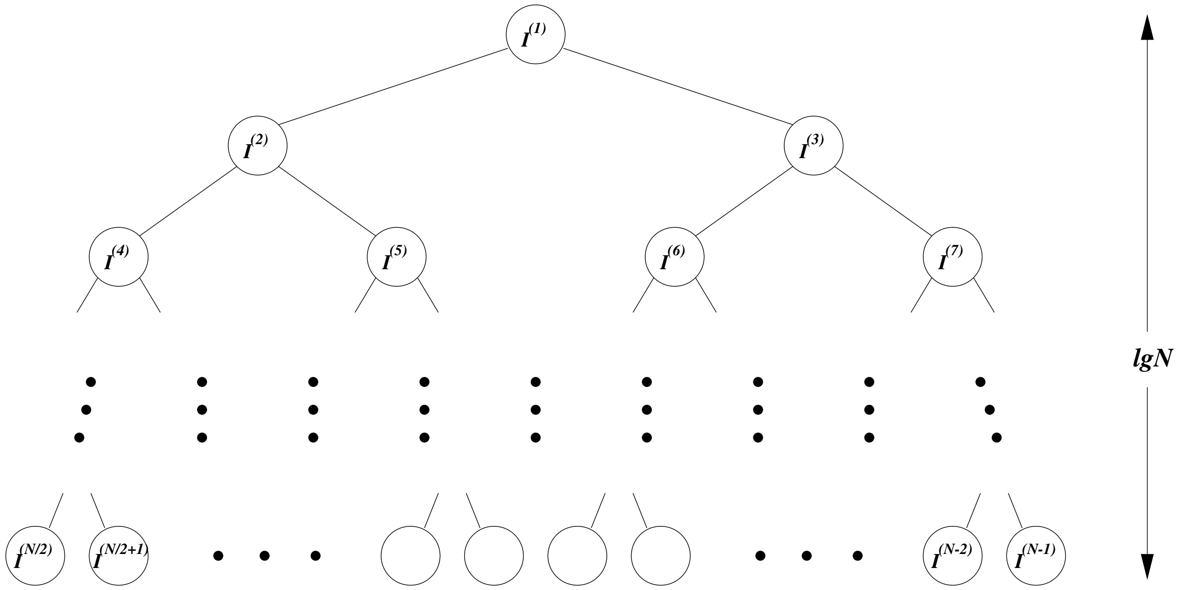

Figure 2.1: The structure of the online instance I^{on}

The ground set $\mathcal{X}^{on}$ of our online instance will consist of the above $N-1$ distinct copies of $\mathcal{X}^{off}$. In order to describe the family of sets F^{on} and the order in which some of the elements are given to the online algorithm, we think of a complete binary tree of depth $\lg N$. We place the $N-1$ copies of I^{off} in the nodes of the tree, one per node, and they are numbered as seen in Figure 2.1. Each of the $N/2$ tree leaves will define M sets of F^{on} in the following manner. Recall that each copy of $\mathcal{X}^{off}$ comes with a family of M sets. Now, the ith set $(1 \leq i \leq M)$ corresponding to a leaf of the tree will be defined to be the union of the ith sets of the $\lg N$ offline instances that sit on the path from the root to the leaf.

Checking the parameters of the online instance, if we denote by n and m the number of elements and sets of I^{on}, we get: $n = N(N-1)$ and $m = NM/2$.

We now turn to specifying which of the elements will be sent to the algorithm and in which order. In general, we will choose one of the $N/2$ leaves and send all the elements of the $\lg N$ offline instances along the path going from top to bottom. This means that $n' = N \lg N$.

The *information theoretic* result which we prove in the following lemma deals with this issue and leads to Corollary 2.3.2, which summarizes the properties of the reduction.

The online instance $I^{on} = \langle \mathcal{X}^{on}, F^{on} \rangle$ is defined by:

- $\mathcal{X}^{on} = \bigcup_{i=1}^{N-1} \mathcal{X}^{(i)off} = \bigcup_{i=1}^{N-1} \bigcup_{l=1}^{N} \{e_l^i\}$
- $F^{on} = \bigcup_{i=N/2}^{N-1} \bigcup_{l=1}^{M} \{\mathbf{set}_l^i\}$, where: $\mathbf{set}_l^i = s_l^i \cup s_l^{\lfloor i/2 \rfloor} \cup s_l^{\lfloor i/4 \rfloor} \cup \ldots \cup s_l^1$
- For each $N/2 \le i \le N-1$ (leaf index) there is a possible input element sequence:

$$\rho_i \ : \quad \underbrace{e_1^1, e_2^1, \ldots, e_N^1}_{\mathcal{X}^{(1)off}} , \ \ldots\ldots\ldots \ , \underbrace{e_1^{\lfloor i/2 \rfloor}, e_2^{\lfloor i/2 \rfloor}, \ldots, e_N^{\lfloor i/2 \rfloor}}_{\mathcal{X}^{(\lfloor i/2 \rfloor)off}} , \ \underbrace{e_1^i, e_2^i, \ldots, e_N^i}_{\mathcal{X}^{(i)off}}$$

Figure 2.2: The online instance I^{on}- formal description

Lemma 2.3.1. *Let DET be any deterministic online algorithm for the online set cover problem and let π be the uniform distribution over the sequences $\rho_{N/2}, \ldots, \rho_{N-1}$. Then,*

$$E_{\pi(i)}\Big[C_{DET}(\rho_i)\Big] \ge \frac{\lg N}{2} \cdot k \ .$$

Proof. The optimal cover size of the original offline instance is k. We fix some arbitrary optimal cover $\mathcal{C}$ for the offline instance. Without loss of generality, $\mathcal{C} = \{s_1, s_2, \ldots, s_k\}$. In order to simplify the analysis of the performance of DET and since we are looking to lower bound it's performance, we will allow DET some prior knowledge (that it may not be able to retrieve, if for instance it is polynomial time, but could retrieve if it is exponential time). This knowledge might only improve the performance of the algorithm. The piece of knowledge is the chosen optimal cover $\mathcal{C}$ of the offline instance. Having this knowledge, we claim that we can restrict the algorithm to choosing sets out of the much smaller variety of sets $\big\{\mathbf{set}_l^i \ : \ l \in [k]\big\}$, that is, only sets that are unions of copies of a set in $\mathcal{C}$. In Claim A.0.3 of Appendix A we verify that this restriction is indeed without loss of generality. Namely, any deterministic algorithm can be replaced by a deterministic algorithm that has the restriction and which performs at least as well as the original algorithm on every single input. In addition, we restrict the algorithm to being *lazy* (i.e. on the arrival of an input element: if the element is covered it doesn't pick a set and if it is uncovered - it picks a single set that covers the element). In Claim A.0.4 of Appendix A we prove that any online algorithm can be assumed to be lazy.

Under these restrictions, the algorithm has exactly k sets corresponding to each leaf of the tree to chose from. These k sets are the unions of the k sets of $\mathcal{C} = \{s_1, s_2, \ldots, s_k\}$ that come from each of the nodes from root to leaf. We call a set that is a union of the copies of s_j a j-type set and we say that it *hits* each of the instances that reside on the path from root to the specific leaf.

For every $i \in [\lg N]$ and $j \in [k]$, we define an indicator random variable x_j^i to equal 1 if during the arrival of the input elements of the ith level instance, DET chooses a j-type set. Clearly, it holds that $C_{DET} \geq \sum_{\substack{i\in[\lg N]\\ j\in[k]}} x_j^i$.

At the arrival of the first level instance, all elements are uncovered and so $x_j^1 = 1$ for every $j \in [k]$. We would like to show that for $i \geq 2$: $\Pr\left[x_j^i = 1\right] = 1/2$ for every $j \in [k]$. We look at the path from the root down to the ith level instance we are interested in. Let s be the last j-type set that the algorithm picked, prior to the arrival of the ith instance. Since $i > 1$ and since the algorithm picked a j-type set on level 1, such a set exists. In addition, from the fact that the algorithm is lazy, this set is the only j-type set picked so far that might hit the ith level instance. By the definition of x_j^i, it holds that: $\Pr\left[x_j^i = 1\right] = \Pr\left[s \text{ doesn't hit the } i\text{th instance}\right]$. Since s is known to hit the $(i-1)$th instance and since each possible path is chosen with equal probability, the path is equally likely to go to each of the the two 'sons' of the $(i-1)$th instance. That is, the above probability equals 1/2.

Now, using the linearity of expectation:

$$\mathrm{E}\left[C_{DET}(I^{on})\right] \geq \sum_{\substack{i\in[\lg N]\\ j\in[k]}} \mathrm{E}\left[x_j^i\right] = \sum_{\substack{i\in[\lg N]\\ j\in[k]}} \Pr\left[x_j^i = 1\right] > \frac{\lg N}{2} \cdot k$$

■

Corollary 2.3.2. *Let A be any online randomized algorithm for Set Cover. There exists a polynomial time reduction from Set Cover to Online Set Cover such that given a Set Cover instance $I^{off} = \langle \mathcal{X}^{off}, F^{off} \rangle$ consisting of N elements and M sets and whose optimal cover size is k, produces an online set cover instance I^{on} with n elements, m sets and element sequences $\rho_{N/2}, \ldots, \rho_{N-1}$, such that:*

- *$n = N(N-1)$ and $m = NM/2$*
- *The optimal offline cover size of each ρ_j is k*

- *The average expected cost of running A on* $\rho_{N/2}, \ldots, \rho_{N-1}$ *is at least* $\frac{\lg N}{2} \cdot k$
- *Furthermore,* $\Pr\left[\sum_{j=N/2}^{N-1} C_A(\rho_j) \;\le\; \frac{N}{2} \cdot \frac{\lg N}{4} \cdot k\right] < 2/3$

Proof. We are left with proving the two last claims of the corollary. Lemma 2.3.1 implies that for any deterministic algorithm DET, it holds that

$$\sum_{j=N/2}^{N-1} C_{DET}(\rho_j) \ge \frac{N}{2} \cdot \frac{\lg N}{2} \cdot k \tag{2.3}$$

Now, Let I be a set of indices and $\{DET_i\}_{i \in I}$ be a corresponding set of deterministic algorithms such that the randomized algorithm A is the deterministic algorithm DET_i with probability λ_i. According to the following calculation, the average expected cost is as we claim.

$$\mathrm{E}\left[\sum_{j=N/2}^{N-1} C_A(\rho_j)\right] = \sum_{j=N/2}^{N-1} \mathrm{E}\left[C_A(\rho_j)\right] = \sum_{j=N/2}^{N-1} \sum_{i \in I} \lambda_i C_{DET_i}(\rho_j) =$$

$$\sum_{i \in I} \lambda_i \sum_{j=N/2}^{N-1} C_{DET_i}(\rho_j) \ge \frac{N}{2} \cdot \frac{\lg N}{2} \cdot k \cdot \sum_{i \in I} \lambda_i = \frac{N}{2} \cdot \frac{\lg N}{2} \cdot k \tag{2.4}$$

It remains to bound the probability of being well under the expectation, as stated in the corollary. As in the proof of Corollary 2.3.2, we assume that every deterministic algorithm is restricted to the family of sets $\left\{\mathbf{set}_l^i \;:\; l \in [k]\right\}$ and that it is lazy. Under these restrictions, any deterministic algorithm cannot choose more than k sets per level (offline instance) and thus we conclude that for every $N/2 \le j \le N-1$: $C_A(\rho_j) \le k \lg N$. Therefore, we can define a new random variable $Y = \frac{N}{2} \cdot k \lg N - \sum_{j=N/2}^{N-1} C_A(\rho_j)$, which is non-negative and apply the Markov inequality on it as follows.

$$\Pr\left[\sum_{j=N/2}^{N-1} C_A(\rho_j) \;\le\; \frac{N}{2} \cdot \frac{\lg N}{4} \cdot k\right] = \Pr\left[Y \ge \frac{3}{2} \cdot \mathrm{E}[Y]\right] < 2/3$$

$\blacksquare$

2.3.2 The Hardness Result

We use the following result of Raz and Safra [RS97], which is based on and improves earlier hardness results of Bellare *et.al.* [BGLR93] and Lund and Yannakakis [LY93].

Theorem 2.3.3 ([RS97]). *There exists a polynomial time reduction from SAT to Set Cover that given a formula φ, produces a Set Cover instance I^{φ} with N elements M sets and optimal solution size OPT, such that:*

- *$M = N^{\alpha}$ for some positive real constant α.*
- *if $\varphi \in SAT$ then $OPT = K(N)$.*
- *if $\varphi \notin SAT$ then $OPT \geq c \cdot \lg N \cdot K(N)$*
  (for some function K that is computable in polynomial time and some constant $0 < c < 1$).

Our hardness results are the following.

Theorem 2.3.4. *There exists a constant $d > 0$ such that the following holds. If there exists a polynomial time randomized algorithm for the online set cover problem with competitive ratio better than $d \cdot \lg n \lg m$, then $NP \subset BPP$.*

Proof. Let A be a polynomial time deterministic randomized for the online set cover problem whose competitive ratio is within $d \cdot \lg n \lg m$, where $d = \frac{c}{40(1+\alpha)}$ (c, α are the constants achieved in theorem 2.3.3).

Reduce SAT to Set Cover according to the reduction described in theorem 2.3.3. We get an instance of Set Cover I_{off} with N elements and $M = N^{\alpha}$ sets. Now, follow the reduction from Set Cover to online set cover from Corollary 2.3.2, to get an instance of online set cover I_{on} with $n = N^2 - N$ elements, $m = NM/2 = N^{1+\alpha}/2$ sets and element sequences $\rho_{N/2}, \ldots, \rho_{N-1}$.

Denote by $OPT(I)$ $\big(OPT(\rho_j)\big)$ the optimal offline cover size of the instance I (sequence ρ_j). We use the convention that a superscript of type *sat* (*unsat*) denotes that the instance (sequence) originated in a satisfiable (unsatisfiable) formula.

If the original formula was satisfiable, by the properties of the reductions we get that $OPT(I^{sat}_{off}) = K(N)$ and therefore, $OPT(\rho^{sat}_j) = K(N)$ for every j. Since A's competitive ratio is within $\frac{c}{40(1+\alpha)} \cdot \lg n \lg m$, $\mathrm{E}\left[\sum_{j=N/2}^{N-1} C_A(\rho^{sat}_j)\right] \leq \frac{N}{2}\Big(\frac{c}{40(1+\alpha)} \cdot \lg n \lg m\Big) \cdot K(N)$ and using the Markov inequality we obtain

$$\Pr\left[\sum_{j=N/2}^{N-1} C_A(\rho_j^{sat}) \geq \frac{N}{2}\Big(\frac{c}{8(1+\alpha)} \cdot \lg n \lg m\Big) \cdot K(N)\right] \leq 1/5\ . \tag{2.5}$$

On the other hand, if the original formula was unsatisfiable, we get that $OPT(I_{off}^{unsat}) \geq (c \cdot \lg N) \cdot K(N)$ and therefore (Corollary 2.3.2):

$$\Pr\left[\sum_{j=N/2}^{N-1} C_A(\rho_j^{unsat}) \leq \frac{N}{2}\frac{\lg N}{4}\big(c \cdot \lg N\big) \cdot K(N)\right] \leq 2/3\ .$$

Substituting $n = N(N-1)$ and $m = N^{1+\alpha}/2$, or even $\lg n < 2\lg N$ and $\lg m < (1+\alpha)\lg N$, we obtain:

$$\Pr\left[\sum_{j=N/2}^{N-1} C_A(\rho_j^{unsat}) \leq \frac{N}{2}\Big(\frac{c}{8(1+\alpha)} \cdot \lg n \lg m\Big) \cdot K(N)\right] \leq 2/3\ . \tag{2.6}$$

Consider the following algorithm. Given a formula φ, run it through the two reductions, to obtain an online set cover instance I, with element sequences $\rho_{N/2}, \ldots, \rho_{N-1}$. Then, run A on each of the $N/2$ sequences of I. Finally, if $\displaystyle\sum_{j=N/2}^{N-1} C_A(\rho_j) \leq \frac{N}{2}\big(\frac{c}{8(1+\alpha)} \cdot \lg n \lg m\big) \cdot K(N)$ say φ is satisfiable and otherwise say it is unsatisfiable. From inequalities (2.5) and (2.6), the acceptance probabilities are: at least 4/5 in the case of a satisfiable formula and at most 2/3 in the case of an unsatisfiable formula. Both reductions are polynomial time and the number $\frac{N}{2}\big(\frac{c}{8(1+\alpha)} \cdot \lg n \lg m\big) \cdot K(N)$ can be computed in polynomial time. Since the acceptance probabilities are bounded away from 3/4, this algorithm is a bounded-probability polynomial time algorithm for SAT. ■

One thing to notice is that the reduction we used is fully deterministic and therefore, if we restrict the claim to deterministic algorithms, the hardness result will be based on the assumption that $P \neq NP$ (rather than the stronger $NP \not\subseteq BPP$ that is needed for randomized algorithms). We formalize this in the next theorem.

Theorem 2.3.5. *There exists a constant $d > 0$ such that the following holds. If there exists a polynomial time deterministic algorithm for the online set cover problem with competitive ratio better than $d \cdot \lg n \lg m$, then P=NP.*

Proof. This proof follows the lines of the previous theorem's proof. Here, we assume the existence of a deterministic algorithm A whose competitive ratio is within $d \cdot \lg n \lg m$.

The first thing we must notice is that both reductions are deterministic and therefore, the process of running both reductions and then running algorithm A on the resulting instance is fully deterministic.

Second, since A is deterministic, we may replace any appearance of $\mathrm{E}[C_A(I)]$ by $C_A(I)$. Therefore, on one hand we have that $\mathrm{E}\left[\sum_{j=N/2}^{N-1} C_A(\rho_j^{sat})\right] \leq \frac{N}{2}\left(\frac{c}{40(1+\alpha)} \cdot \lg n \lg m\right) \cdot K(N)$ and on the other, using the third statement of corollary 2.3.2,

$$\mathrm{E}\left[\sum_{j=N/2}^{N-1} C_A(\rho_j^{unsat})\right] \geq \frac{N}{2}\frac{\lg N}{2}\left(c \cdot \lg N\right) \cdot K(N) \geq \frac{N}{2}\left(\frac{c}{4(1+\alpha)} \cdot \lg n \lg m\right) \cdot K(N) \ .$$

This implies that the deterministic polynomial time algorithm we construct for SAT is always correct, as needed.

■

Chapter 3

Algorithms and Upper Bounds

3.1 The Case $C_{OPT} = 1$

Let $\mathcal{X}$ be the ground set of n elements and $\mathcal{F}$ - is the family of m subsets of $\mathcal{X}$. In this section we assume that $C_{OPT} = 1$, namely, that if $e_1, e_2, \ldots, e_{n'}$ is the *sequence* of the elements that is given by the adversary, then if we denote by $\mathcal{X}'$ the set $\mathcal{X}' = \{e_1, e_2, \ldots, e_{n'}\}$, then there is a set $s \in \mathcal{F}$ such that $\mathcal{X}' \subseteq s$. We will denote: $\mathcal{F}^j = \left\{s \in \mathcal{F} \;:\; \{e_1, e_2, \ldots, e_j\} \subseteq s\right\}$. The following relationship clearly holds:

$$\phi \neq \mathcal{F}^{n'} \subseteq \mathcal{F}^{n'-1} \subseteq \ldots \subseteq \mathcal{F}^2 \subseteq \mathcal{F}^1 \subseteq \mathcal{F}$$

Algorithm *1UNIFORM*

◇ On the arrival of an element e_j :

1. If the prefix $[e_j] = e_1, e_2, \ldots, e_j$ is already covered by a single set (a set from $\mathcal{F}^j$), do nothing and wait for the arrival of the next element.
2. Else, choose uniformly at random a single set s from $\mathcal{F}^j$ and add it to the cover.

Theorem 3.1.1. *For any set-cover instance whose optimal cover consists of a single set:*

$$E[C_{1UNIFORM}] \leq \mathcal{H}_m < \ln m + 1$$

Proof. The idea here is to first characterize the worst kind of input for the *1UNIFORM* algorithm and then compute the expected cover size it computes for such an instance.

Given the set (sequence) $\mathcal{X}' = \{e_1, \ldots, e_{n'}\}$ of input elements, for every $1 \leq i \leq n'$ we denote: $[e_i] = \{e_1, e_2, \ldots, e_i\}$. We start by putting some restrictions on the instance, which with respect to this algorithm, can only cause an increase in the resulting cover size. First, we assume that every set $s \in S$ covers at least one element of $\mathcal{X}'$ (otherwise, let m' be the number of sets in $\mathcal{F}$ that do intersect $\mathcal{X}'$ and we will get a bound of $\mathcal{H}_{m'}$, which is even smaller). Second, we assume that $|F^{n'}| = 1$ (there is only one optimal set), otherwise we could remove some of the elements off any extra optimal set, only making it harder for the algorithm. Third, we assume that the algorithm is lazy (claim A.0.4 of Appendix A allows us to make this assumption).

We have the relationship $\phi \neq \mathcal{F}^{n'} \subseteq \mathcal{F}^{n'-1} \subseteq \ldots \subseteq \mathcal{F}^2 \subseteq \mathcal{F}^1 = S$ and if we denote $l_i = |\mathcal{F}^i|$ for all $1 \leq i \leq n'$, we have: $1 = l_{n'} \leq l_{n'-1} \leq \ldots \leq l_2 \leq l_1 = |\mathcal{F}| = m$.

If we denote by A_i the event that *1UNIFORM* chose a set on the arrival of e_i (i.e. the element e_i is uncovered on its arrival), by the linearity of expectation, the following holds.

$$\mathrm{E}[C_{1UNIFORM}] = \sum_{i=1}^{n'} \Pr[A_i] \tag{3.1}$$

Trivially, $\Pr[A_1] = 1$. For any $2 \leq i \leq n'$: denote by i' the index of the last element $e_{i'}$ ($i' < i$) which was uncovered on its arrival. Let s' be the set chosen of the arrival of $e_{i'}$. From the above definition of the index i': $s' \in F^{i-1}$. Whether the event A_i happens is fully determined by whether $s' \in F^i$. The probability of this happening is exactly $\Pr[A_i] = \frac{|\mathcal{F}^{i-1}| - |\mathcal{F}^i|}{|\mathcal{F}^{i-1}|} = \frac{l_{i-1} - l_i}{l_{i-1}}$.

Plugging this into (3.1), we get:

$$\mathrm{E}[C_{1UNIFORM}] = \sum_{i=1}^{n'} \Pr[A_i] = 1 + \sum_{i=2}^{n'} \frac{l_{i-1} - l_i}{l_{i-1}} \tag{3.2}$$

This expression is maximal exactly when $l_{i-1} = l_i + 1$ for all $1 < i \leq n'$, which also implies (from the assumption that $l_{n'} = 1$ and $l_1 = m$) that $n' = m$. This is true, since on one hand, if $m < n'$: there exists some i such that $L_{i-1} = l_i$. Adding to the instance a new set $s \in \mathcal{F}^{i-1} - \mathcal{F}^i$ would increase the expression in (3.2). On the other hand, similarly, if $m > n'$: there exists some i such that $L_{i-1} = l_i + t$ for some integer $t > 1$ and in this case - removing from the instance a set $s \in \mathcal{F}^{i-1} - \mathcal{F}^i$ would also increase the expression in (3.2). So, we can assume that $l_i = m - i + 1$. We finally get:

$$\mathrm{E}[C_{1UNIFORM}] \quad = \quad 1 + \sum_{i=2}^{m} \frac{1}{l_{i-1}} = 1 + \sum_{i=2}^{m} \frac{1}{m-i+2} = 1 + \sum_{i=2}^{m} \frac{1}{i} = \mathcal{H}_m$$

■

3.2 An $O(k \cdot \lg \frac{m}{k})$-competitive Randomized Algorithm

The series of algorithms we discuss in this section are designed to work on instances for which the optimal solution is of some size k, which is known to the algorithm.

If we had (for every value of k) an algorithm ALG_k that is $f(k)$-competitive, we could (under some reasonable assumptions regarding f) derive an $8f(k)$-competitive algorithm for the general case, by using the technique of doubling. The algorithm will be the following.

Algorithm *General*

1. Let $k = 1$.
2. While the cover size is at most $k \cdot f(k)$ - keep running ALG_k on the input.
3. Set $k = 2k$ and return to step 2.

We note that since the algorithms of the type ALG_k that we design might run in time exponential in k, algorithm *General* might also run in time exponential in k.

Theorem 3.2.1. *If for any value of k there exists an algorithm ALG_k that is $f(k)$-competitive against instances with $C_{OPT} = k$ (where $f(k)$ is some function that is monotone increasing in k and which satisfies $f(\alpha k) \leq \alpha f(k)$, for any $\alpha \geq 0$), then algorithm General is an $8f(k)$-competitive randomized algorithm for the general online set cover problem.*

Proof. Assume we run algorithm *General* on an instance I with $C_{OPT} = k$. When we shift from algorithm ALG_{2^i} to algorithm $ALG_{2^{i+1}}$, since all the input elements till the current element have already been covered (and payed for) by previous algorithms, we think as if the input started at the current element, we aren't allowed to chose any previously chosen sets and we discard incoming elements that have been covered by previous chosen sets. This leaves the new algorithm $ALG_{2^{i+1}}$ with a new instance, whose optimal solution is not larger than the optimal solution of the original instance. It is clear from here that the algorithm finishes at the latest during the running of Alg_{2^j}, where: j is the minimal integer such that $2^j \geq k$.

The total number of chosen elements is at most: $\sum_{i=1}^{j} 2^i \cdot f(2^i) \leq 2^{j+1} f(2^j) \leq 4kf(2k) \leq 8kf(k)$ and we get the needed ratio.

■

A consequence of Theorem (3.2.1) is that we may allow any algorithm for the online set cover problem to have an additional piece of knowledge about the instance at the beginning of its running - that is, the size of an optimum solution. This additional information can improve the competitive ratio of the algorithm by at most a constant factor.

3.2.1 Algorithm *UNIFORM*

Let k denote the size of an optimal solution to the instance that is given to the algorithm. Define a k-set to be a set of k sets out of $\mathcal{F}$. We say that a k-set covers an element if the union of the k sets covers the element. This algorithm is a generalization of the *1UNIFORM* algorithm and we note that it runs in time exponential in k.

Algorithm *UNIFORM*

◇ On the arrival of an element e_j :

1. If the prefix $[e_j] = e_1, e_2, \ldots, e_j$ is already covered by a single k-set, do nothing and wait for the arrival of the next element.
2. Else, choose uniformly at random a k-set out of the k-sets that cover the prefix $[e_j] = e_1, \ldots, e_j$ and add the k sets it contains to the cover.

Theorem 3.2.2. *The UNIFORM algorithm is $O\left(k \lg \frac{m}{k}\right)$-competitive on instances $I = \langle \mathcal{X}, S \rangle$ for which $C_{OPT} = k$.*

Proof. The proof is by reduction to the proof of algorithm *1UNIFORM*. Look at the new instance $I^k = \langle \mathcal{X}, F^k \rangle$, where F^k is the set of all k-sets of $\mathcal{F}$, where each k-set is viewed as a single set that contains all the sets in the union. $C_{OPT}(I) \leq k$ guaranties that

$C_{OPT}(I^k) = 1$. We notice that the algorithm *UNIFORM* on I acts precisely like *1UNIFORM* does on I^k. Denote: $M = |F^k| = \binom{m}{k}$. Since *1UNIFORM* is $\ln M$-competitive and since $\ln M = \ln \binom{m}{k} = \Theta\left(k \lg \frac{m}{k}\right)$, $O\left(k \lg \frac{m}{k}\right)$ k-sets will be chosen. Therefore, UNIFORM will pick such a number of k-sets, which turns out to be $O\left(k^2 \lg \frac{m}{k}\right)$ sets. This proves the desired competitiveness.

■

3.2.2 An instance on which this bound is realized

Description

We begin with describing the elements of the instance, which will be given in two phases. Phase I consists of n_1 elements, which are all given to the algorithm in an arbitrary order. Phase II consists of a 2-dimensional array of elements, with k rows and $k \ln M$ columns. Each of the array elements will be given to the algorithm - column after column, from left to right, going top to bottom within each column. We now proceed to describing the sets of the instance. We call a set (i, j)-associated if out of the array elements it covers exactly the j first elements of the i'th row and no other array elements. Every set of this instance will be (i, j)-associated for some $i \in [k]$ and $j \in [k \ln M]$.

For each $i \in [k]$ and $j \in [k \ln M]$ we define $M \cdot (1 - \frac{1}{k})^{j-1}$ (i, j)-associated sets. Looking at a single row from left to right: M sets are associated with the first element and the decrease from column to column is by a factor of $(1 - \frac{1}{k})$ (We took the maximal number of columns possible under these circumstances). These are the instance's sets, but it remains to define how they cover the elements of phase I.

For each $j \in [k \ln M]$ (corresponding to a column of the array), create a partition of phase I's n_1 elements into k parts. We desire that the union of any k parts that do not all belong to a single partition, will not entirely cover the n_1 elements of phase I.

A set of such partitions can be constructed probabilistically as follows. For each $j \in [k \ln M]$ (corresponding to a column of the array), create a partition $P_1^j, P_2^j, \ldots, P_k^j$ of phase I's n_1 elements uniformly at random into k equally sized parts (n_1 can be taken a multiple of k).

On one hand, Pr[k sets, that don't belong to a single partition, cover phase I] $\leq$

$$\Pr[k \text{ sets, that all except one belong to a single partition, cover phase I}] = \binom{n_1}{k}^{-1} \leq \left(\frac{k}{n_1}\right)^k.$$

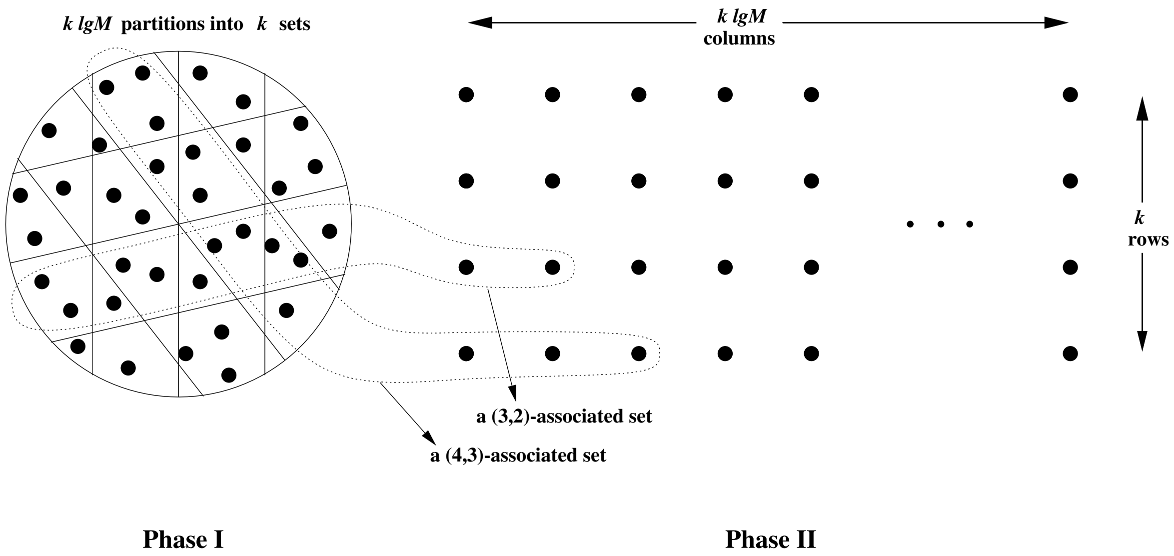

Figure 3.1: The Array Instance - A bad instance for algorithm *UNIFORM*

On the other, the number of possible choices of k parts out of $k \ln M$ k-partitions is at most $\binom{k^2 \ln M}{k}$, which can be bounded by $(ek \cdot \ln M)^k$.

Therefore, the expected number of k parts that aren't in a single partition but cover phase I is at most $\left(\frac{ek^2 \ln M}{n_1}\right)^k$. Taking n_1 larger than $ek^2 \ln M$ will bring this to less than 1. We take a choice of sets that realizes this expectation.

We also give a deterministic construction for such a set of partitions. Let $[k^2 \ln M]$ be the set of elements of phase I. Then, we define the $k \ln M$ k-partitions $\{P_1^j, \ldots, P_k^j\}_{j=1}^{k \ln M}$ in the following way:

$$P_i^j = [k \ln M] + (j-1) + (i-1) \cdot k \qquad (\text{mod } k \ln M) \qquad \big(\text{where } [s] + t \overset{d}{=} \{1+t, \ldots, s+t\}\big).$$

Finally, each (i,j)-associated set will cover the elements in P_i^j, but no other element from phase I.

If we denote by n and m the number of elements and sets correspondingly, we have that: $n = \Theta(k^2 \lg M)$ and $m = k \cdot M \cdot \sum_{i=1}^{k \lg M} \left(1 - \frac{1}{k}\right)^i = k \cdot M \cdot \left(k - \frac{k}{M}\right) = \theta(k^2 M)$. In addition, k is the optimal cover size.

Theorem 3.2.3. *The competitive ratio of UNIFORM is at least* $\Omega\Big(k \lg \frac{m}{k}\Big)$.

Proof. We analyze the performance of UNIFORM on the instance described above. Namely, we show that it is expected to pick at least $\Omega\big(k^2 \lg \frac{m}{k}\big)$ sets.

Phase I was designed especially so that when an element of a column j $(j \geq 2)$ of phase II arrives, the only k-sets that cover the entire prefix (these are the k-sets that the algorithm chooses from at random) are those, where each set is associated to a different row, but all are associated with a single column, one of $\{j, j+1, \ldots, k \ln M\}$. We will call such a k-set that is associated with column l or with one of the following columns an l-coordinated k-set. The reason for the above is that if the k sets are associated to different columns, they will fail to cover the elements of phase I and on the other hand if there isn't one set associated with each of the k rows, the elements of one of the rows will not be covered on previous columns.

To summarize the above, using the new terminology: During phase II, when an element of column j $(j \geq 2)$ arrives and the prefix isn't covered by any k-set from the current cover, the algorithm chooses at random a k-set from the j-coordinated k-sets.

To continue the proof, we will need the following lemma, whose technical proof we give in Appendix A.

Lemma 3.2.4. *At some stage of the incoming elements of phase II, an l-coordinated k-set will be chosen (for some l such that the k-set isn't $(l+1)$-coordinated) and no elements that lie to the right of the lth column will be covered. Furthermore, this happens with constant probability for some $l = o\big(k \lg M\big)$.*

Based on this lemma, from the lth column and on, the algorithm will pick a new k-set only on the arrival of the first element of a column, if it is uncovered. The number of columns that the algorithm has to deal with is $k \ln M - l = \Omega(k \lg M)$.

If we denote by A_j the event that the algorithm chose a k-set on the arrival of the first element of the jth column, by the linearity of expectation and since we assume we are in this situation with constant probability c, the following holds.

$$\mathrm{E}[C_{Uniform}] \geq c \cdot k \cdot \sum_{j=l}^{k \ln M} \Pr[A_j] \tag{3.3}$$

For any $l \leq j \leq k \ln M$: denote by j' the index of the last row $(j' < j)$ whose first element was uncovered on its arrival. Let $K_s = \{s_1, \ldots, s_k\}$ be the k-set chosen on the arrival. Whether the event A_j happens is fully determined by whether K_s contains the first element of the jth column. Since K_s is a $(j-1)$-coordinated k-set, the probability of this happening is exactly the fraction of j-coordinated k-sets out of the $(j-1)$-coordinated k-sets. The number of j-coordinated k-sets is $\left(\frac{M}{k} \cdot (1 - \frac{1}{k})^{j-1}\right)^k \approx \left(\frac{M}{k}\right)^k \cdot e^{1-j}$. Therefore, the above fraction, meaning $\Pr[A_j]$ is approximately $1/e$. Summing this up, we get:

$$\mathrm{E}[C_{Uniform}] = \Omega\left(k^2 \lg M\right) = \Omega\left(k^2 \lg \frac{m}{k}\right)$$

■

3.3 An $O(\lg n' \lg m)$-competitive Polynomial Time Randomized Algorithm

3.3.1 Algorithm *RandSC*

The following algorithm is a *randomized* algorithm for the online set cover problem. We note that it is a *polynomial time* algorithm, as is the deterministic algorithm of Alon *et.al.*. Each set s will be assigned a weight ω_s and similarly, each element e_j will have a weight ω_j, which is defined to be, at any stage, the sum of the weights of the sets containing that element. We denote by $\mathcal{S}_j$ the collection of sets that contain the element e_j and by $\mathcal{C}$- the family of sets which forms the constructed cover.

Algorithm *RandSC*

◇ Initialization: $\mathcal{C} = \phi$ and for each $s \in \mathcal{F}$: $\omega_s = max_k \left\{2^{-k} \ : \ 2^{-k} < \frac{1}{m}\right\}$.

◇ Pick each $s \in \mathcal{F}$ with probability w_s to $\mathcal{C}$.

◇ On the arrival of the jth element e_j:

1. If e_j is covered, do nothing and wait for the arrival of a new element.
2. Else (e_j is uncovered), if $\omega_j > \lg j$, add to $\mathcal{C}$ an arbitrary single set from $\mathcal{S}_j$, do not update any weights and wait for the arrival of a new element.
3. Else (e_j is uncovered and $\omega_j \leq \lg j$), perform the following weight augmentation:
   (a) For each set $s \in \mathcal{S}_j$, if $\omega_s < 1$ double its weight: $\omega_s = 2\omega_s$.
   (b) For each set $s \in \mathcal{S}_j$, independently, add s to $\mathcal{C}$ with probability w_s.
   (c) If e_j isn't yet covered (no set was added in step (b)), return to step 2.

◇ Return the cover $\mathcal{C}$.

Theorem 3.3.1. *The RandSC algorithm is $O(\lg n' \lg m)$-competitive against an oblivious adversary (where $n' = |\mathcal{X}'|$ =actual number of elements given by the adversary).*

Proof. We show that the *RandSC* algorithm constructs a *feasible* cover $\mathcal{C}$ of $\mathcal{X}'$, whose expected size is bounded by $O(\lg n' \lg m) \cdot C_{OPT}$.

First, it is clear that the cover $\mathcal{C}$ which is constructed by *RandSC* is feasible, since when a new element j arrives, it has either been previously covered, or else it will be covered in the current series of weight augmentations. Second, we will calculate the expected size of the cover $\mathcal{C}$. Let $\mathcal{C}'$ be an optimal offline-produced cover. For each set $s \in \mathcal{C}'$, its weight w_s is initialized to be $max_k \left\{2^{-k} \;:\; 2^{-k} < \frac{1}{m}\right\}$ and never exceeds 2, since once it exceeds 1, it is surely added to the cover. On the other hand, in every weight augmentation, at least one set in $\mathcal{C}'$ participates, so we conclude that if we denote by *augs* the number of weight augmentations that *RandSC* performs, we have that:

$$augs \leq (\lg m + 1) \cdot C_{OPT} \tag{3.4}$$

In addition, since in any of the augmentations that follow the arrival of e_j, the element's weight ω_j never exceeds $2 \lg j$, we can bound the expected number of sets added in each augmentation:

$$\text{E}[\# \text{ of added sets during an augmentation that follows the arrival of some element } e_j] \leq \sum_{s \in \mathcal{S}_j} 2\omega_s = 2\omega_j < 4 \lg j \leq 4 \lg n' \tag{3.5}$$

Denote by X the number of sets added to the cover at the beginning of the algorithm (line 2). Clearly,

$$\text{E}[X] = \sum_{s \in \mathcal{F}} \omega_s \leq 1. \tag{3.6}$$

We denote by Y the r.v. that counts the total number of sets that are added to $\mathcal{C}$ by step 2 of the algorithm and similarly - denote by Z the r.v. which counts the total number of sets that are added to $\mathcal{C}$ by step (b). We clearly have that $C_{RandSC} = X + Y + Z$.

Bounding the expectation of Y is based on the low probability of adding a set to $\mathcal{C}$ in step 2.

$$\begin{aligned}
\text{E}[Y] &\leq \sum_{j=1}^{n'} \Pr\left[j \text{ is uncovered and } \omega_j > \lg j\right] \\
&= \sum_{j=1}^{n'} \prod_{s \in \mathcal{S}_j} (1 - \omega_s) \leq \sum_{j=1}^{n'} \left(1 - \frac{\omega_j}{|\mathcal{S}_j|}\right)^{|\mathcal{S}_j|} \\
&\leq \sum_{j=1}^{n'} e^{-\omega_j} < \sum_{j=1}^{n'} e^{-\lg j} = \sum_{j=1}^{n'} \frac{1}{j} = \mathcal{H}_{n'} < \ln n' + 1
\end{aligned} \tag{3.7}$$

Similarly, using linearity of expectation:

$$\begin{aligned} \mathrm{E}[Z] &= \sum_{j=1}^{augs} \mathrm{E}[\#\text{ of sets added in any augmentation}] \\ &\leq \sum_{j=1}^{augs} (4\lg n') = augs \cdot 4\lg n' = O(\lg n' \lg m) \cdot C_{OPT} \end{aligned} \tag{3.8}$$

The first inequality follows from (3.5) and the second inequality follows from (3.4). Combining (3.6), (3.7) and (3.8) by the linearity of expectation, we finally get:

$$\mathrm{E}[C_{RandSC}] = \mathrm{E}[X] + \mathrm{E}[Y] + \mathrm{E}[Z] = O(\lg n' \lg m) \cdot C_{OPT}$$

■

3.3.2 An instance on which this bound is realized

Description

We look at the Set-Cover instance $\mathcal{I} = \langle \mathcal{X}, \mathcal{F} \rangle$ where:

$$\mathcal{X} = \underbrace{x_1^1, x_2^1, \ldots, x_m^1}_{\mathcal{X}^1}, \underbrace{x_1^2, x_2^2, \ldots, x_m^2}_{\mathcal{X}^2}, \ldots, \underbrace{x_1^k, x_2^k, \ldots, x_m^k}_{\mathcal{X}^k}$$

where $k = \Theta(\lg m)$ and the family $\mathcal{F}$ of sets is defined by: $\mathcal{F} = \mathcal{F}^1 \cup \{s_1\}$ where $\mathcal{F}^1$ hierarchically decomposes into:

$$\mathcal{F}^1 \supseteq \mathcal{F}^2 \supseteq \ldots \supseteq \mathcal{F}^k$$

where: $|\mathcal{F}^{i+1}| = |\mathcal{F}^i|/9$, which leaves us with $\mathcal{F}^k = \{s_0\}$ (assuming that $|\mathcal{F}|$ is a power of 9) being a single set and the containment of elements is defined as follows:

- Each set in $\mathcal{F}^1$ contains a random half of the elements in $\mathcal{X}^1$.
- Each set in $\mathcal{F}^2$ contains (in addition) the 'same' random half of the elements in $\mathcal{X}^2$.
- Each set in $\mathcal{F}^i$ contains (in addition) the 'same' random half of the elements in $\mathcal{X}^i$. [1]
- By these definitions, the single set s_0 in $\mathcal{F}^k$ contains a random half of the complete set of elements $\mathcal{X}$. In order to have $C_{OPT} = 2$, we define an extra complementary set: $s_1 = \mathcal{X} - s_0$.

[1] For example, if a set in $\mathcal{F}^i$ contains the elements $x_1^1, x_2^1, \ldots, x_{m/2}^1$ of the first block $\mathcal{X}^1$, then it will contain in addition the elements $x_1^j, x_2^j, \ldots, x_{m/2}^j$ for $1 < j \leq i$.

Summarizing all the sizes: $n = |\mathcal{X}| = km$, $m = |\mathcal{F}|$ and $k = \Theta(\lg m)$.

Theorem 3.3.2. *The competitive ratio of RandSC is at least* $\Omega(\lg n \lg m)$.

We give a formal proof to this theorem in Appendix A. The following is the intuition to its correctness.

Proof Idea. We analyze the performance of *RandSC* on the instance $\mathcal{I}$ described above. The expected number of sets needed in order to cover block $\mathcal{X}^1$ of the input elements is $\lg m$ (since every set covers a random half of the uncovered elements). This number of sets is chosen when $\omega(F^1)$ reaches $\lg m$ (since the weight of a family is the expected number of sets chosen from it). The algorithm cannot tell if a set in F^1 is also in F^2 and keeping in mind that $|F^2| = |F^1|/9$, we get that $\omega(F^2)$ reaches $\lg m/9$, $\omega(F^3)$ reaches $\lg m/81$ and so on.

In order to cover the block $\mathcal{X}^2$, we need $\omega(F^2)$ to reach $\lg m$. Continuing in this manner we get that $8 \lg m/9$ sets are expected to be chosen in each of the rounds $2, 3, \ldots, k$. So, $\mathrm{E}[C_{RandSC}] \geq 8 \lg^2 m/9 = \Omega(\lg n \lg m)$, whereas $C_{OPT} = 2$ (since $\{s_0, s_1\}$ is a cover) and we obtain the desired bound.

■

3.4 An Instance on which Both Upper Bounds are Realized

We include this example, since it seems to be quite general. Both randomized algorithms we gave here meet their upper bounds on it and many other randomized algorithms that we analyzed, failed to do well on this instance or on slight variations of it.

Description

Our instance consists of a 2-dimensional array of elements together with sets that partially cover them. The array has k rows and $k \ln M$ columns. Denote by $e(i, j)$ the element located at the intersection of the ith row and the jth column $\big(1 \leq i \leq k$ and $1 \leq j \leq k \ln M\big)$. Every one of the array's elements will be given to the algorithm - column after column, from left to right, going from top to bottom within each column.

We now proceed to describing the sets. We say that a *set* is *j-associated* if it covers exactly one element from each of the first j columns and no element from the proceeding columns. We say that a k-set (a family of k sets)is *j-associated* if each of its sets is *j-associated* and if the union of its k sets entirely covers the first j columns. Notice (since the number of sets in a k-set equals the number of rows) that this happens if and only if the k-set consists of k

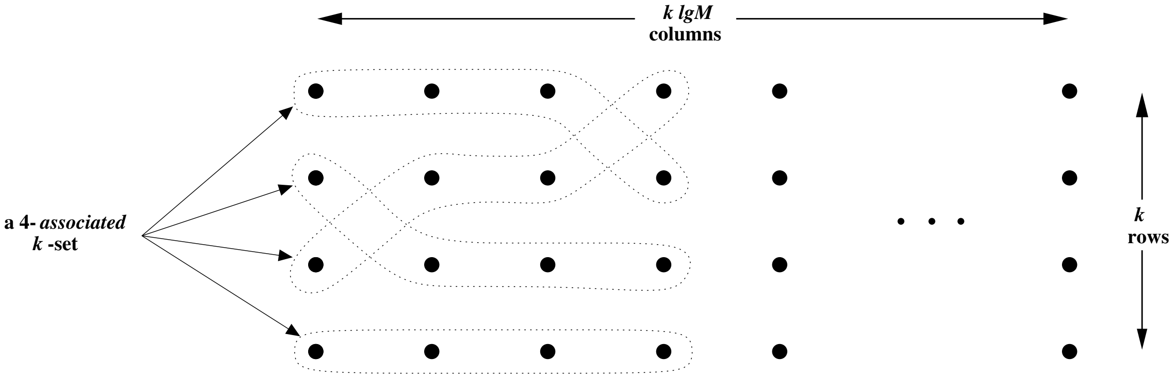

Figure 3.2: A *4-associated* k-set that corresponds to the 3 permutations (for columns 2,3,4): $\{1,3,2,4\},\{1,3,2,4\},\{2,3,1,4\}$.

pairwise disjoint *j-associated* sets. In this view, a *j-associated* k-set can be defined by a series of $j-1$ permutations of $[k]$, which we denote by $\{\pi_l\}_{l=2}^{j}$. Namely, when looking at the lth column ($2 \le l \le j$) - the rth set (which is the set that covers the rth element of column 1) of the k-set covers the element $e(\pi_l(r), l)$.

For the jth column we independently pick $j-1$ permutations of $[k]$, $\{\pi_l\}_{l=2}^{j}$, uniformly at random. This implicitly defines a *j-associated* k-set, as noted above. We create $M \cdot \left(1-\frac{1}{k}\right)^{j-1}$ copies of each of these k-sets. Any choice of one copy of each of the sets clearly forms a *j-associated* k-set. We see that these $kM \cdot \left(1-\frac{1}{k}\right)^{j-1}$ sets form $M^k \cdot \left(1-\frac{1}{k}\right)^{k(j-1)} \approx M^k \cdot e^{-(j-1)}$ *j-associated* families of sets.

It is evident that the number of associated *sets* decreases from one column to the next by a factor of $1-1/k$, while the number of associated *k-sets* decreases by a factor of approximately $1/e$ (an exponential rate). Notice also that we took the maximal number of columns possible since the number of sets and k-sets associated with the first column is kM and M^k correspondingly and both these numbers decrease right down to being constants towards the last column.

Before we analyze the performance of both algorithms on our example - lets summarize the sizes of the instance. First, $C_{OPT}=k$, since there is more than one family of k sets that is associated with the last column. Second, the number of sets is:
$m = k \cdot M \cdot \sum_{j=1}^{k \lg M}\left(1-\frac{1}{k}\right) = k \cdot M \cdot k\left(1-\frac{1}{M}\right) = \Theta(k^2 M)$. Third, the number of elements is: $n = k^2 \lg M$.

The proofs for the following two propositions are presented in Appendix A.

Proposition 3.4.1. *Algorithm UNIFORM will pick an expected number of* $\Omega(k^2 \cdot \lg \frac{m}{k})$ *sets when running on the instance described above and therefore its competitive ratio is at least* $\Omega(k \cdot \lg \frac{m}{k})$.

Proposition 3.4.2. *Algorithm RandSC will pick an expected number of* $\Omega(k \lg n \lg m)$ *sets when running on the instance described above and therefore its competitive ratio is at least* $\Omega(\lg n \lg m)$.

Bibliography

[AAA+03] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. The online set cover problem. In *Proc. of the annual ACM Symposium on the Theory of Computing (STOC)*, pages 100–105, 2003.

[AAFL96] 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 *Proc. of the annual ACM Symposium on the Theory of Computing (STOC)*, 1996.

[Alb97] S. Albers. Better bounds for online scheduling. In *Proc. of the annual ACM Symposium on the Theory of Computing (STOC)*, pages 130–139, 1997.

[AS00] N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley New York, 2000.

[BDBK+94] S. Ben-David, A. Borodin, Richard Karp, Gabor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11:2–14, 1 1994.

[BFKV95] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *J. of Computer and System Sciences*, 51(3):359–366, 1995.

[BFL96] Y. Bartal, A. Fiat, and S. Leonardi. Lower bounds for on-line graph problems with application to on-line circuit and optical routing. In *Proc. of the annual ACM Symposium on the Theory of Computing (STOC)*, pages 531–540, 1996.

[BGLR93] M. Bellare, S. Goldwasser, C. Lund, and A. Russel. Efficient probabilistically checkable proofs and applications to approximations. In *Proc. of the annual ACM Symposium on the Theory of Computing (STOC)*, pages 294–304, 1993.

[BLS92] A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task system. *J. of ACM*, 39(4):745–763, 1992.

[Chv79] V. Chvátal. A Greedy Heuristic for the Set Covering Problem. *OMEGA*, 4(3):233–235, 1979.

[CW98] J. Csirik and G. J. Woeginger. On-line packing and covering problems. In *Online Algorithms, The State of the Art*, pages 147–177. Springer-Verlag, 1998.

[Fei98] U. Feige. A threshold of ln$n$ for approximating set cover. In *J. of the ACM, 45(4)*, pages 634–652, 1998.

[FKL+91] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *J. of Algorithms*, 12(4):685–699, 1991.

[HS92] M. M. Halldórsson and M. Szegedy. Lower bounds for on-line graph coloring. In *Proc. of the third annual ACM Symposium on Discrete Algorithms (SODA)*, pages 211–216, 1992.

[Joh74] D. S. Johnson. Approximation algorithms for combinatorial problems. In *J. Computer System Science, 9*, pages 256–278, 1974.

[KPT96] D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *J. of Algorithms*, 20(2):400–430, 1996.

[KVV90] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In ACM, editor, *Proc. of the Annual ACM Symposium on Theory of Computing (STOC)*, pages 352–358, 1990.

[Loo46] L.H. Loomis. On a theorem of von neumann. In *Proc. of the National Academy of Sciences of the U.S.A.*, volume 32, pages 213–215, 1946.

[Lov75] L. Lovász. On the ratio of the optimal rational and integral covers. In *J. of Discrete Mathematics 13*, pages 383–390, 1975.

[LST89] L. Lovász, M. Saks, and W. T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *J. of Discrete Math.*, 75(1-3):319–325, 1989.

[LY93] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In *Proc. of the annual ACM Symposium on the Theory of Computing (STOC)*, pages 286–293, 1993.

[MR97] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1997.

[MS91] L.A McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[MvN44] O. Morgenstern and J. von Neumann. Theory of games and economic behavior. Princeton University Press, 1944.

[RS97] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proc. of the Annual ACM Symposium on Theory of Computing (STOC)*, pages 475–484, 1997.

[Sla96] P. Slavík. A tight analysis of the greedy algorithm for set cover. In *Proc. of the annual ACM Symposium on the Theory of Computing (STOC)*, pages 435–441, 22–24 May 1996.

[Sri] A. Srinivasan. Improved approximation guarantees of packing and covering problems. In *Proc. of the annual ACM Symposium on the Theory of Computing (STOC)*, pages 268–276.

[ST85] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *J. of Communication ACM*, 28(2):202–208, 1985.

[Yao77] A. C-C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. of the annual ACM Symposium on the Theory of Computing (STOC)*, pages 222–227. IEEE, 1977.

Appendix A

Technical Proofs

Claim A.0.3. *For every deterministic algorithm DET, there exists a deterministic algorithm DET' which is restricted to the family of sets $\left\{\boldsymbol{set}_l^i\ :\ l \in [k]\right\}$, and which chooses at most the number of sets that DET does on any of the $N/2$ possible input element sequences of I^{on}.*

Proof. The instance I^{on} has the set of elements $\mathcal{X}^{on} = \{e_l^i\ :\ l \in [N],\ i \in [N-1]\}$ and the family of sets $\left\{\mathbf{set}_l^i\ :\ l \in [M],\ N/2 \leq i \leq N-1\right\}$. We remind here that the upper index always corresponds to the node number in the tree and the lower index corresponds to the element/set number in the instance I^{off}.

Let ρ_t be an element sequence that DET is running on, for some $N/2 \leq t \leq N-1$. This element sequence contains all the elements of each of the $\lg N$ offline instances that lie on the path from the root to the leaf with index t. When a set $\mathbf{set}_l^i$ is picked, we say it hits each of the $\lg N$ instances that lie on the corresponding path.

We describe which sets DET' picks during the level i instance $I^{(j_i)off}$. We can assume that during each of the $\lg N$ instances, any algorithm will pick sets only on the arrival of the instance's first element, since it knows that all the rest of the instance's elements will follow.

The algorithm's description appears in figure A.1.

First, we have to show that the algorithm is well defined, in the sense that on each instance $I^{(j_i)off}$ it holds that $k - q \leq r$. We prove this by induction on the level i of the instance j_i.

For $i = 1$, (the root instance $I^{(1)off}$), $q = 0$. Γ_1 is the family of sets that DET used to cover $I^{(1)off}$. Since k is the optimal size of a cover for the offline instance, indeed $k \leq |\Gamma_1| = r$.

Assume this condition held for the first $i-1$ levels and the algorithm reaches the ith instance $I^{(j_i)off}$ without getting stuck. Notice in the algorithm that the upper indices of Λ_{j_i} and Γ_{j_i} are identical, meaning, that at each stage, both algorithms pick the exact same

Algorithm DET'

$\diamond$ On the arrival of the first element of the instance $I^{(j_i)off}$:

- Let $\Delta_{j_i} = \{\mathbf{set}_{\varphi_1}^{\psi_1}, \ldots, \mathbf{set}_{\varphi_q}^{\psi_q}\}$ be the past elements that DET' has picked and which hit the current instance $I^{(j_i)off}$. $\big(\varphi_l \in [k]$ and $N/2 \leq \psi_l \leq N-1\big)$
- Denote $\{\varphi_{q+1}, \varphi_{q+2}, \ldots, \varphi_k\} = [k] - \{\varphi_1, \varphi_2, \ldots, \varphi_q\}$.
- Let $\Gamma_{j_i} = \{\mathbf{set}_{\delta_1}^{\epsilon_1}, \ldots, \mathbf{set}_{\delta_r}^{\epsilon_r}\}$ be the sets that DET chose during the instance $I^{(j_i)off}$.
- The r sets that will be chosen are

$$\Lambda_{j_i} = \{\mathbf{set}_{\varphi_{q+1}}^{\epsilon_1}, \ldots, \mathbf{set}_{\varphi_k}^{\epsilon_{k-q}}, \mathbf{lowest}^{\epsilon_{k-q+1}}, \ldots, \mathbf{lowest}^{\epsilon_r}\}$$

  where $\mathbf{lowest}^i$ is the lowest index set out of $\{\mathbf{set}_1^i, \ldots, \mathbf{set}_k^i\}$ that DET' hasn't yet chosen or the empty set in the case that all the sets have been chosen.

Figure A.1: Description of the restricted algorithm DET'

number of sets corresponding to each and each leaf (unless DET' has already picked all k relevant sets of the leaf). Since this holds for all the previous $i-1$ instances, when the algorithms reach the ith level, they both have hit the ith instance exactly the same number (q) of times (unless DET hit more than k times and in this case, $q = k$ and the inequality holds). Again, since k is the optimal size cover, DET will need at least $k - q$ sets to cover the instance.

Second, the cover that DET' produces is legal, since at the beginning of each offline instance, it completes the family of k-sets, which cover the instance.

Finally, it is clear that in a stage that DET picks r sets, DET' picks at most r sets.

$\blacksquare$

Claim A.0.4. *Any algorithm for the online set cover problem can be replaced by a lazy algorithm, which does at least as well as the original algorithm on every single instance. We define an algorithm to be lazy if it doesn't pick sets that are unnecessary at the time. Formally, if a covered element arrives no set is picked and if an uncovered element arrives - a single set that covers it is picked.*

Proof. Let ALG be an arbitrary algorithm. A new algorithm ALG' will run ALG in the background. When ALG picks a set that covers an uncovered element - ALG' picks the same set and when ALG picks any other set - ALG' keeps it marked in memory. Now, if it happens that an element that arrives is covered by ALG but not covered by ALG', ALG' simply picks one of the marked sets in memory that covers the element. The family of sets that ALG' picks is clearly a subset of that of ALG. Nevertheless, ALG' is a lazy algorithm.

■

Lemma 3.2.4 *At some stage of the incoming elements of phase II, an l-coordinated k-set will be chosen (for some l such that the k-set isn't $(l+1)$-coordinated) and no elements that lie to the right of the lth column will be covered. Furthermore, this happens with constant probability for some $l = o\big(k \lg M\big)$.*

Proof. Denote by t the number of sets that the algorithm picks during phase I. Each of these sets is associated with some column of phase II. We will first bound the rightmost column that any of these sets could be associated with.

In the analysis that will give us this bound, we look at the t sets as if they were picked uniformly at random. The reason that we can assume this is that it can be seen that the more the sets are 'correlated' (in the sense that many sets come from the same partitions) the less far right they will reach in the array of phase II.

The way we look at the process is the following. First, we know that when we randomly throw t balls into a set of bins standing in a line, the rightmost ball is expected to have a fraction of $1 - 1/(t+1)$ of the bins to its right. In our case, we want to estimate how many columns lie to the right of the rightmost column that any of the t sets is associated with. However, in our case the number of sets associated to a column decreases by a factor of $(1 - 1/k)$ from column to column, moving from right to left.

In order to use the balls and bins model, we need to take a set s that has a fraction of $1 - 1/(t+1)$ of the sets that are higher associated (associated with columns that are to the right of the column that s is associated with) and check which column it is associated with. The index of this column is the expected value we are interested in bounding.

$$\frac{(\#\text{ of elements that are associated with at most the } (k\ln t)\text{'th column})}{(\text{total number of sets})} =$$

$$\frac{kM \cdot \sum_{j=0}^{k\ln t}\left(1-\frac{1}{k}\right)^j}{kM \cdot \sum_{j=0}^{k\ln M}\left(1-\frac{1}{k}\right)^j} = \frac{\left(1-\left(1-\frac{1}{k}\right)^{k\ln t}\right)\Big/\frac{1}{k}}{\left(1-\left(1-\frac{1}{k}\right)^{k\ln M}\right)\Big/\frac{1}{k}} \approx \frac{\left(1-\frac{1}{t}\right)}{\left(1-\frac{1}{M}\right)} > \left(1-\frac{1}{t}\right) \tag{A.1}$$

This inequality means, according to the above explanation, that when picking t random sets, the far most to the right a set will reach is expected to be the $k \ln t$ th column.

In our case, trivially $t = O\big(k^3 \lg M\big)$, since $n_1 = O\big(k^2 \lg M\big)$ and at the worst case k sets will be chosen on the arrival of each element. Therefore, the rightmost column that a set will be associated with is expected to be at most $k \ln t = O\big(k \ln(k^3 \lg M)\big) = o\big(k \lg M\big)$.

Denote the index of this column by l. By the Markov inequality, with constant probability: $l = o\big(k \lg M\big)$.

Now, we saw that when the algorithm is in phase II it only picks coordinated k-sets. Let j_1 be the last column before (including) l, on which a coordinated k-set was chosen. This chosen k-set is a random l-coordinated k-set. As before, since the number of $(l+1)$-coordinated k-sets is smaller than the number of l-coordinated k-sets by a factor of $1/e$, the chosen k-set, will not be $(l+1)$-coordinated with probability $1 - 1/e$. In addition, no elements that lie to the right of the lth column will be covered, as required.

■

Theorem 3.3.2. *The competitive ratio of RandSC is at least* $\Omega\left(\lg n \lg m\right)$.

Proof. We analyze the performance of *RandSC* on the instance $\mathcal{I}$ from subsection 3.3. In that instance there exists a cover of size 2 $\left(\{s_0, s_1\}\right)$, but we show that *RandSC* will pick an expected number of $\Omega\left(\lg^2 m\right)$ sets, which gives, taking in account that $\lg m = \Theta(\lg n)$, the desired result.

We probabilistically construct the instance, by letting each set cover a random half of some of the blocks, as described in subsection 3.3. In the calculations that follow, the probability is taken both over the random construction of the sets and over the random choices of the algorithm. Once we do the calculations, it will be clear that there exists a fixed setting (construction) of the instance's sets, such that these probabilities hold. Then, we will obtain that under this construction, the algorithm is expected to behave in a certain way.

We begin by proving that certain 'bad' events occur with negligible probability. Call a family of sets *regular* if it doesn't contain neither of the sets s_0, s_1. We remind that the number of blocks was taken to be $k = \lg_9 m < \lg m$. According to Claim A.0.4, it will suffice to lower bound the lazy version of algorithm *RandSC*.

$$\Pr\left[\begin{matrix}\text{There exists a block which } \mathit{RandSC} \text{ hits}\\ \text{more than } 2\lg m \text{ times before covering}\end{matrix}\right] \le \lg m \cdot \Pr\left[\begin{matrix}\text{A random regular family of } 2\lg m\\ \text{sets doesn't cover the block } \mathcal{X}_1\end{matrix}\right] \le$$

$$m \lg m \cdot \Pr\left[\begin{matrix}\text{A random regular family of } 2\lg m \text{ sets}\\ \text{doesn't cover a given element in } \mathcal{X}_1\end{matrix}\right] = \frac{m\lg m}{2^{2\lg m}} = \lg m/m \xrightarrow[m\to\infty]{} 0 \quad \text{(A.2)}$$

Under the condition that this low probability event does not occur, we bound the probability of the following event.

$$\Pr\left[\begin{matrix}\mathit{RandSC} \text{ will pick a special}\\ \text{set during the first } k/2 \text{ blocks}\end{matrix}\right] \le \lg m \cdot \Pr\left[\begin{matrix}\mathit{RandSC} \text{ will pick a special set during}\\ \text{the arrival of elements of the block } \mathcal{X}_{k/2}\end{matrix}\right] \le$$

$$\lg m \cdot \Pr\left[\begin{matrix}\text{A special set is chosen, when choosing}\\ 2\lg m \text{ random sets out of } \Theta(\sqrt{m}) \text{ sets}\end{matrix}\right] \le 2\lg m \cdot \frac{2\lg m}{\sqrt{m}} \xrightarrow[m\to\infty]{} 0 \quad \text{(A.3)}$$

In addition,

$$\Pr\Big[\exists \text{ a regular family of } \lg m/2 \text{ sets that covers } \mathcal{X}_1\Big] \le$$

$$\binom{m}{\lg m/2} \cdot Pr\Big[\text{a random regular family of } \lg m/2 \text{ sets covers } \mathcal{X}_1\Big] =$$

$$\binom{m}{\lg m/2} \cdot \left(1 - \frac{1}{2^{\lg m/2}}\right)^m \le \binom{m}{\lg m} \cdot e^{-\sqrt{m}} \le m^{\lg m} \cdot e^{-\sqrt{m}} \xrightarrow[m\to\infty]{} 0 \quad \text{(A.4)}$$

We assume that these three previous low probability events do not occur. This won't asymptotically affect the expected performance of the algorithm.

Take a *minimal* sized regular family of sets, L, that covers the elements of the first block $\mathcal{X}^1$. By our assumption, $|L| \ge \lg m/2$. Denote by L' the family of sets in L, restricted to the elements of $\mathcal{X}^1$. Now, by an argument that is similar to that of claim A.0.3, we do not limit the power of *RandSC* by restricting it to sets whose restriction to $\mathcal{X}^1$ is in L'.

If we denote $L' = \{s_1, s_2, \ldots, s_{|L|}\}$, we may define for every $i \in [k/2]$ and $j \in [|L|]$ an indicator random variable x_j^i to equal 1 if during the arrival of the input elements of $\mathcal{X}^i$, *RandSC* chooses a set whose restriction to $\mathcal{X}^1$ is s_j. Clearly, we have $C_{\mathit{RandSC}} \ge \sum_{\substack{i\in[k/2]\\ j\in[|L|]}} x_j^i$.

At the arrival of the first block, all elements are uncovered and so $x_j^1 = 1$ for every $j \in [|L|]$. We would like to show that for $i \ge 2$: $\Pr\left[x_j^i = 1\right] = 1/9$ for every $j \in [k/2]$. Call a set whose restriction to $\mathcal{X}^1$ is s_j a j-type set. Every set that *RandSC* is allowed to pick is a j-type set for some $j \in [|L|]$ and furthermore, by the definitions, for each $j \in [|L|]$ and

$i \in [k/2]$, *RandSC* must pick at least one j-type set that hits $\mathcal{X}^i$. We look at the last j-type set that the algorithm picked, prior to the arrival of $\mathcal{X}^i$. Since $i > 1$ and since the algorithm picked a j-type set on level 1, such a set exists. In addition, from the fact that the algorithm is lazy, this set is the only j-type set picked so far that might hit the ith level instance. By the definition of x_j^i, it holds that: $\Pr\left[x_j^i = 1\right] = \Pr\left[s \text{ doesn't hit } \mathcal{X}^i\right]$. Since s is known to hit $\mathcal{X}^{i-1}$ and since each set in $\mathcal{F}^{i-1}$ is in $\mathcal{F}^i$ w.p. 1/9, the above probability equals 1/9.

Now, using the linearity of expectation:

$$\mathrm{E}\left[C_{RandSC}\right] \geq \sum_{\substack{i\in[k/2]\\ j\in[|L|]}} \mathrm{E}\left[x_j^i\right] = \sum_{\substack{i\in[k/2]\\ j\in[|L|]}} \Pr\left[x_j^i = 1\right] = \Omega\big(\lg^2 m\big)$$

■

Proposition 3.4.1. *Algorithm UNIFORM will pick an expected number of* $\Omega(k^2 \cdot \lg \frac{m}{k})$ *sets when running on the instance described above and therefore its competitive ratio is at least* $\Omega(k \cdot \lg \frac{m}{k})$.

Proof. In order to prove this proposition, we follow the lines of the proof of Theorem 3.2.3. In both cases there is the 2 dimensional array of elements and it is of same size. In the other result, we proved (in lemma 3.2.4) that from some column and on, according to a probabilistic construction, only coordinated k-sets will be chosen. This is based on the special structure of the instance that assures that, when reaching the array phase, every k-set that covers the entire prefix of elements must be coordinated. In our case, we speak of associated k-sets instead of coordinated k-sets and we show that the same thing happens, not at the beginning of the array, but after a constant fraction of the columns, say $j = \frac{1}{10} k \lg M$. Then, from that stage on, every j-associated k-set that UNIFORM picks advances it by an expected constant number of columns. Thus, it is expected to pick at least a number of $k \cdot \frac{1}{10} k \lg M = \Omega(k^2 \cdot \lg \frac{m}{k})$ sets.

Call a k-set a *bad* k-set if it isn't associated and if it covers the first j columns of the array.

$$\Pr[k \text{ sets, that aren't associated, cover the first } j \text{ columns}] \leq$$

$$\Pr\begin{bmatrix} k \text{ sets, that all except one are part of an} \\ \text{associated } k\text{-set, cover the first} j \text{ columns} \end{bmatrix} = \left(\frac{1}{k}\right)^j = 1/k^{\frac{1}{10}k \lg M} = 1/e^{\lg M \lg k}. \quad \text{(A.5)}$$

Since the number of k-sets that cover the first column is $(kM)^k$, using the union bound, we obtain that the expected number of bad k-sets is at most:

$$(kM)^k / k^{\frac{1}{10}k \lg M} = k^{k + \frac{k \lg M}{\lg k}} / k^{\frac{1}{10}k \lg M} \xrightarrow[m\to\infty]{} 0$$

We fix a construction that realizes this expectation.

■

Proposition 3.4.2. *Algorithm RandSC will pick an expected number of* $\Omega(k \lg n \lg m)$ *sets when running on the instance described above and therefore its competitive ratio is at least* $\Omega(\lg n \lg m)$.

Proof. This proof is informal. The key to the analysis is the observation, that at any stage of the input elements, each of the sets looks the same with respect to the containment of previous elements. Namely, for each j, when the jth column elements are arriving, the weight of all the sets that are associated with the jth column or with a higher column is expected to be identical. This happens, since each set that reaches column j, hits one random element of each of the j first columns.

The weight of a set at a certain stage, equals its probability of being chosen for the cover at that stage. Since these probabilities are expected to be similar, what the algorithm actually does, is very close to picking sets uniformly at random. It might pick many sets at the arrival of the single element - if it is of high weight. An algorithm that picks a single set, uniformly at random, on the arrival of any uncovered set, is expected to do better than *RandSC* on this instance. We will use the performance of such an algorithm as an upper bound on the performance of RandSC, namely, we will analyze its performance instead of that of RandSC.

From equality (A.1) and the comment that followed it, we can conclude that a random chosen set is expected to be associated with a column that is an order of k columns ahead of the current one. Suppose that the algorithm picks t sets per column, meaning that t out of the k elements of a column arrive uncovered. By the previous comment, each column is hit, prior to the arrival of its elements an expected order of kt times. These kt hits fall randomly on the elements of the column. When throwing p balls into q bins, uniformly at random, the expected number of empty bins at the end of the process is $q \cdot e^{-p/q}$. In our case: there are kt balls, thrown into k bins, with t empty bins expected. We can therefore write: $t = k \cdot e^{-kt/k}$, which can be written as: $t = \lg(k/t) = \lg(k/\lg(k/t)) = \ldots = \Theta(\lg k)$. Therefore, the expected size of a resulting cover is t times the number of columns: $\lg k \cdot k \lg M$. We remind here that the number of elements and sets were: $n = k^2 \lg M$ and $m = k^2 M$. There, we must chose $k = O(\sqrt{n})$. We take k to be maximal under this restriction and then, since $\lg k = \Theta(\lg n)$, we get that the number of set chosen is $\Theta(k \lg n \lg m)$.

■