

Bucket Game with Applications to Set Multicover and Dynamic Page Migration*

Marcin Bienkowski¹ and Jarosław Byrka²

¹ International Graduate School of Dynamic Intelligent Systems,
University of Paderborn, Germany
young@upb.de

² Centrum voor Wiskunde en Informatica,
Kruislaan 413, NL-1098 SJ Amsterdam, Netherlands
J.Byrka@cwi.nl

Abstract. We present a simple two-person *Bucket Game*, based on throwing balls into buckets, and we discuss possible players' strategies. We use these strategies to create an approximation algorithm for a generalization of the well known Set Cover problem, where we need to cover each element by at least k sets. Furthermore, we apply these strategies to construct a randomized algorithm for Dynamic Page Migration problem achieving the optimal competitive ratio against an oblivious adversary.

1 Introduction

In this paper we present a simple two-player *Bucket Game*. In this game we have a set of n initially empty buckets, an infinite set of balls and a constant parameter $0 < c < 1$. In each turn, player A associates arbitrarily a non-negative weight w_i with each bucket i . We can easily extend the notion of weight to sets of buckets, i.e. the weight of a set X is the sum of weights of all buckets from X (the total weight is the sum of weights of all n buckets). On the basis of weights $\{w_i\}$, player B chooses a subset of buckets, whose weight is at least a fraction c of the total weight, and throws a ball into each bucket from this subset. The goal of player A is to fill each bucket with at least T balls for a given threshold T in as few rounds as possible, whereas the goal of player B is to postpone it.

One of the most straightforward questions that arises is: “Assuming the optimal strategy of player B , how fast can player A achieve the given threshold T ?”.

* Extended abstract. The full version of this paper is available under <http://www.hni.upb.de/publikationen/>.

¹ Partially supported by DFG-Sonderforschungsbereich 376 “Massive Parallelität: Algorithmen Entwurfsmethoden Anwendungen”, and by the Future and Emerging Technologies programme of the EU under EU Contract 001907 DELIS “Dynamically Evolving, Large Scale Information Systems”.

² Supported by the EU Marie Curie Research Training Network ADONET, Contract No MRTN-CT-2003-504438.

In this paper we address this issue, proving tight bounds in Sect. 2. The trivial answers to this question which we present are $\mathcal{O}(T \cdot \log n)$ and $\Omega(T + \log n)$. We develop a simple *Exponential Balancing* technique, which constructively yields an algorithm for player A . This algorithm is guaranteed to fill all the buckets to the given threshold T in $\mathcal{O}(T + \log n)$ rounds. All logs in the paper are to base 2, unless stated otherwise.

Although the Bucket Game might be interesting itself, the main contribution of our paper is applying the Exponential Balancing scheme to improve approximation and competitive ratios of Set Multicover and Dynamic Page Migration problems, respectively. We present the problems and summarize our results, separately, in the two following subsections.

1.1 Set Cover, Set Multicover and k -SetCover

The Set Cover problem is defined as follows. Given a collection C of subsets of S , such that $\bigcup_{s_i \in C} s_i = S$, find a collection $C' \subseteq C$, such that $\bigcup_{s_i \in C'} s_i = S$ and $|C'|$ is minimal.

The Set Cover problem is NP-complete. Moreover, it was proved by Raz and Safra [12] that the existence of an $(c \cdot \log n)$ -approximation algorithm for Set Cover with $c < 1$ would imply that $\text{NP} \subseteq \text{DTIME}(n^{\log \log n})$. On the other hand, Johnson [8] and Lovász [9] showed two different algorithms, both of which approximate Set Cover within a factor of $H_n = \sum_{i=1}^n \frac{1}{i} = \Theta(\log n)$.

A natural generalization of Set Cover is the Set Multicover problem, where each element x of S needs to be covered by at least l_x sets, and each subset $s_i \in C$ may be used arbitrary number of times. This problem was addressed by Rajagopalan and Vazirani in [11], where an H_n -approximation algorithm was presented. We will, however, concentrate on the particular case where each element must be covered by at least k subsets (i.e. $l_x = k$ for all $x \in S$). We call this problem k -SetCover. A similar problem – a variant where each set may be used at most once (known as Constrained Set Multicover) – was recently shown in [3] to have applications to reverse engineering of protein and gene networks.

Our Contribution. We propose an algorithm for the k -SetCover problem, that in polynomial time produces a k -cover with $\mathcal{O}((k + \log n) \cdot c^*)$ sets where c^* is a number of sets in the optimal solution to the classical Set Cover problem. In other words, c^* is the cost of the optimal solution of the original input instance with $k = 1$. Furthermore, our algorithm can be extended without loss of approximation guarantee to handle a weighted version of k -SetCover, i.e. the one with different costs charged for using particular sets in the cover.

Our algorithm is based on a reduction from Set Cover to Uncapacitated Facility Location (UFL) problem. The reduction is similar to the one used by Guha and Khuller [7] to prove a lower bound on the approximation factor for UFL.

Although we do not improve the approximation ratio for k -SetCover achieved by the greedy algorithm of Vazirani, we believe the result is worth of interest. It relates the cost of the computed solution to c^* which is for some instances $\Omega(k)$ times smaller than the optimal solution to k -SetCover. We give an example of instances for which our algorithm computes $\Omega(\log n)$ times cheaper solutions.

1.2 Dynamic Page Migration

The *Dynamic Page Migration* (DPM) problem [4,5] arises in a network of n processors (nodes) v_1, v_2, \dots, v_n , which share one indivisible memory page (shared variable) of size D . This variable is stored in the local memory of one of these processors, initially at v_1 . The processors are placed in a metric space (\mathcal{X}, d) , i.e. the distances between the points from \mathcal{X} are given by the metric d .

We assume discrete time steps $t = 1, 2, \dots$. At the beginning of the step the adversary may move each node by at most a constant distance Δ , i.e. for any node v_i , its positions $p_{t-1}(v_i)$ and $p_t(v_i)$ in two consecutive time steps cannot be too far apart, $d(p_{t-1}(v_i), p_t(v_i)) \leq \Delta$. A tuple describing the positions of all the nodes in time t is called *configuration at time step t* and is denoted by \mathcal{C}_t .

After moving nodes, the adversary chooses one node, denoted by σ_t , which wants to access (read or write) a single unit of data from a page. If the page is stored in its local memory, then such a transaction is free. Otherwise, the node has to send a request to the processor holding the page, say v^* , and appropriate data is sent back. This incurs a cost, which is defined to be $c_t(\sigma_t, v^*) := d_t(\sigma_t, v^*) + 1$. The function $d_t(v_a, v_b)$ denotes the distance in step t between any two nodes v_a and v_b , i.e. $d_t(v_a, v_b) := d(p_t(v_a), p_t(v_b))$.

To avoid the problem of maintaining consistency among multiple copies of the page, the model allows only one copy of the page to be stored within the network. After serving the request, the algorithm may decide to migrate the page to another processor. The migration cost between two nodes v_a and v_b is equal to $D \cdot c_t(v_a, v_b)$.

The goal is to decide, online, when and where to move the page in order to exploit the locality of the request, and *minimize the total cost of communication* for all possible pairs of sequences of requests (σ_t) and network changes (\mathcal{C}_t) .

We consider only online algorithms, i.e. the ones which make decision in step t solely on the basis of the initial part of the input up to step t . To analyze the performance of an online algorithm ALG we use competitive analysis [13]. We say that a randomized algorithm ALG is c -competitive, if for all input sequences $(\mathcal{C}_t, \sigma_t)$ it holds $\mathbf{E}[C_{\text{ALG}}((\mathcal{C}_t, \sigma_t))] \leq c \cdot C_{\text{OPT}}((\mathcal{C}_t, \sigma_t))$, where the expected value is taken over all random choices made by an algorithm. $C_{\text{ALG}}((\mathcal{C}_t, \sigma_t))$ and $C_{\text{OPT}}((\mathcal{C}_t, \sigma_t))$ are the cost of ALG and the *optimal offline solution*, respectively, on the input sequence $(\mathcal{C}_t, \sigma_t)$. The factor c is called the *competitive ratio* of the algorithm. In this paper we consider only oblivious adversaries [2], which have no access to the random bits used by the algorithm.

Related Work. The case in which network is static, i.e. $\mathcal{C}_t = \mathcal{C}_{t+1}$ for all time steps t and the constant overhead for communication is not present, called *Page Migration*, had been introduced in [6]. The best known algorithms (deterministic and randomized), achieving constant competitive ratios for general topologies, were presented in [1,14].

For the DPM problem Bienkowski, Dynia, and Korzeniowski gave in [4] a deterministic, $\mathcal{O}(\min\{\sqrt{D} \cdot n, D, \lambda\})$ -competitive algorithm MARK, where λ is the maximum distance between any pair of nodes occurring during runtime. This result is up to a constant factor optimal due to the matching lower bound for

any randomized algorithm playing against an adaptive-online adversary given in [5]. In [4] it was also shown that a direct randomization of MARK yields the algorithm R-MARK which is $\mathcal{O}(\min\{\sqrt{D} \cdot \log n, D, \lambda\})$ -competitive against an oblivious adversary. The best known lower bound for this case, given in [4], is $\Omega(\min\{\sqrt{D \cdot \log n}, D^{2/3}, \lambda\})$.

Our Contribution. In Sect. 4 we partially close the gap mentioned above. We use Exponential Balancing technique to approximate the node holding page of the optimal algorithm by an accurate probability distribution over all nodes. We prove that our algorithm is $\mathcal{O}(\sqrt{D \cdot \log n})$ -competitive. Since it can be combined with trivial $\mathcal{O}(D)$ and $\mathcal{O}(\lambda)$ algorithms from [5], we get an algorithm which is $\mathcal{O}(\min\{\sqrt{D \cdot \log n}, D, \lambda\})$ -competitive against an oblivious adversary.

2 Bucket Game

In this section we formally define a two-player *Bucket Game* and discuss possible strategies for each player.

Definition 1 (Bucket Game). *Assume we have a set of n buckets, which are initially empty, numbered from 1 to n and let $[n] := \{1, 2, \dots, n\}$. We also have an infinite set of balls. For any $i \in [n]$, let c_i be the current number of balls in bucket i . Let $0 < c < 1$ be any fixed constant. The Bucket Game is played in rounds by two players A and B . Each round of the game is defined as follows.*

- *Player A defines a sequence of non-negative weights $\{w_i\}_{i=1}^n$ and shows it to player B .*
- *Player B chooses some subset $X \subseteq [n]$ of buckets, s.t. $\sum_{i \in X} w_i \geq c \cdot \sum_{i=1}^n w_i$, and throws exactly one ball into each bucket from X .*

The game ends when each of the buckets contains at least T balls (i.e. $c_i \geq T$ for all $i \in [n]$). The goal of player A is to minimize the number of rounds, while B wants to play as long as possible.

Let us make the following simple observations. First of all, to throw at least one ball into each bucket (i.e. for the case $T = 1$), $\mathcal{O}(\log n)$ rounds are sufficient. Player A simply defines $w_i = 1$ for empty buckets and $w_i = 0$ for non-empty ones. Then in each round at least a fraction c of empty buckets gets a ball. Hence, after at most $\log_{1/(1-c)} n$ rounds there is no empty bucket left. Thus, to fill each bucket to the threshold T , player A can repeat this scheme T times, which yields an upper bound of $\mathcal{O}(T \cdot \log n)$.

Second, for any T , there exists a player B strategy which prevents finishing the game in less than $\Omega(T + \log n)$ rounds. Since each bucket may get at most one ball per round, the number of rounds cannot be smaller than T . On the other hand, suppose that in every round B chooses the subset with the smallest number of empty buckets. With this strategy, in the i -th round, at most $\lceil c \cdot e_i \rceil$ of empty buckets get a ball, where e_i is the number of empty buckets at the beginning of the i -th round. Thus $\Omega(\log n)$ rounds are also necessary.

Surprisingly, there is a simple player A strategy, which we call *Exponential Balancing*, and which asymptotically matches the above-mentioned lower bound.

Theorem 1. *For $T = \lfloor \log_2 n \rfloor$, there exists a player A strategy which guarantees finishing the game in $\mathcal{O}(\log n)$ rounds.*

Proof. In each round A defines $w_i = \frac{n}{2^{c_i}}$. We call w_i a *value of a bucket*, and define the total value of the game as $\mathcal{W} = \sum_{i \in [n]} w_i$.

Initially, all buckets are empty, $w_i = n$ for all i . Hence, the initial value of the game is n^2 . In any round each bucket “offers” to player B a half of its current value for putting a ball into this bucket. According to the rules of the game, B must collect at least a fraction c of this offered value. Thus, in every round, the game loses at least a fraction $\frac{1}{2} \cdot c$ of its value. If \mathcal{W} and \mathcal{W}' denote the game values in two consecutive rounds, then $\mathcal{W}' \leq (1 - c/2) \cdot \mathcal{W}$. If in some round $\mathcal{W} \leq 1$, then the threshold T is reached and the game ends. Precisely, $\mathcal{W} = \sum_i \frac{n}{2^{c_i}} \leq 1$ implies $\frac{n}{2^{c_i}} \leq 1$ for all $i \in [n]$, and thus $c_i \geq \log n$ for all $i \in [n]$. It remains to observe that the value of the game can be reduced from n^2 to 1 in at most $\log_{1/(1-c/2)} n^2 = \frac{2}{1-\log_2(2-c)} \cdot \log_2 n$ rounds. \square

If threshold T is larger than $\lfloor \log_2 n \rfloor$, then player A may act as if he was playing $\lceil \frac{T}{\lfloor \log_2 n \rfloor} \rceil$ times a game with threshold $\lfloor \log_2 n \rfloor$. By Theorem 1, each of these sub-games lasts $\mathcal{O}(\log n)$ rounds, and thus the whole game ends after at most $\mathcal{O}(T)$ rounds. Thus, we get the following.

Corollary 1. *For any T , there exists a player A strategy which guarantees finishing the game in $\mathcal{O}(T + \log n)$ rounds.*

3 Application to Set Multicover

In this section we present a new approximation algorithm for the k -SetCover problem, the modification of the Set Multicover defined in Sect. 1.1. Our algorithm uses an approximation algorithm for the Uncapacitated Facility Location (UFL) problem as a subroutine.

In the following, we say that A is a λ -approximation algorithm for a minimization problem P , if for any instance of the problem P it produces, in polynomial time, a solution with a cost at most λ times higher than the cost of an optimal solution.

Uncapacitated Facility Location (UFL) Problem. In the UFL problem we are given a set \mathcal{F} of n_f facilities and a set \mathcal{C} of n_c cities. For every facility $i \in \mathcal{F}$, a non-negative number f_i denotes the *opening cost* of the facility. Furthermore, for every city $j \in \mathcal{C}$ and facility $i \in \mathcal{F}$, c_{ij} is a *connection cost* between facility i and city j . The goal is to open a subset of the facilities $\mathcal{F}' \subseteq \mathcal{F}$, and connect each city to an open facility so that the total cost is minimized.

The UFL problem is NP-complete, and MAX SNP-hard (see [7]). A UFL instance is *metric* if its *connection cost* function satisfies the *triangle inequality* (i.e. $c_{ij} \leq c_{ik} + c_{kj}$ for any $i, j, k \in \mathcal{C} \cup \mathcal{F}$). There are several approximation

algorithms for the metric UFL problem, the currently best one achieving the approximation ratio of 1.52 [10].

Guha and Khuller [7] have proved by a reduction from Set Cover that there is no polynomial time λ -approximation algorithm for metric UFL with $\lambda < 1.463$, unless $NP \subseteq DTIME(n^{\log \log n})$. Another of their results was a $(1.463 \dots + \epsilon)$ -approximation algorithm for the case when all the connection costs are either 1 or 3. We use this algorithm in our construction.

Computing Partial Set-Covers. First, we address a problem of computing *partial set-covers* for a given instance of the Set Cover problem. By a *set-cover* we mean a feasible solution to an instance of the Set Cover problem (i.e. a family $C' \subset C$ covering every element of S), whereas a *partial set-cover* is a family $C'' \subset C$ that covers at least a certain fraction of elements of S .

We put weights on elements to indicate that covering certain elements is more important than covering others. We would like to know how much we can cover with at most $k \cdot c^*$ sets, for a constant $k \in \mathcal{R}_+$ and c^* denoting the number of sets in an optimal set-cover.

Let λ_0 be the approximation factor of an algorithm for the metric UFL problem with connection costs 1 or 3 (By [7], we may choose $\lambda_0 \approx 1.463$).

Lemma 1. *For all $k > \frac{2\lambda_0-1}{2-\lambda_0}$ there exists an algorithm, that given $((S, C), c^*, w)$ (where (S, C) is an instance of the Set Cover problem, c^* is the number of subsets in its optimal solution, and $w : S \rightarrow N_+$ is a weight function on the elements of S) runs in polynomial time in $\sum_{x \in S} w(x)$, outputs a partial set-cover C_p with the number of sets $c_p \leq k \cdot c^*$, and the total weight of elements not covered by C_p is at most $\frac{k(\lambda_0-1)}{2k-2\lambda_0} \cdot \sum_{x \in S} w(x)$.*

To prove this lemma, we use a reduction of a Set Cover instance to a UFL instance with distances 1 and 3 and an approximation algorithm for the UFL problem. The core of the reduction and the algorithm were proposed by Guha and Khuller [7]. We slightly extended the original reduction to encode the elements' weights into quantities of groups of cities representing particular elements. The complete proof of Lemma 1 can be found in the full version of the paper.

In Lemma 1 we bounded the fraction of uncovered weight of elements by $\frac{k(\lambda_0-1)}{2k-2\lambda_0}$, when $k \cdot c^*$ subsets are used. Suppose we want to cover certain fraction c of elements' weight and wonder how many sets do we need to use. We may consider the uncovered weight fraction $1-c = \frac{k(\lambda_0-1)}{2k-2\lambda_0}$ and conclude the following.

Corollary 2. *For any $0 < c < \frac{3-\lambda_0}{2} \approx 0.768$ there exists an algorithm that, in polynomial time, computes a partial set-cover that covers at least a fraction c of elements' weight, using at most $\frac{2\lambda_0 \cdot (1-c)}{3-\lambda_0-2c} \cdot c^*$ sets. We call this algorithm Set-UFL $_c$.*

3.1 Approximation Algorithm for k -SetCover

Now we combine Corollary 2 with Theorem 1 to present an algorithm for the k -SetCover problem. First, we present an algorithm A_1 (see Fig. 1), solving the

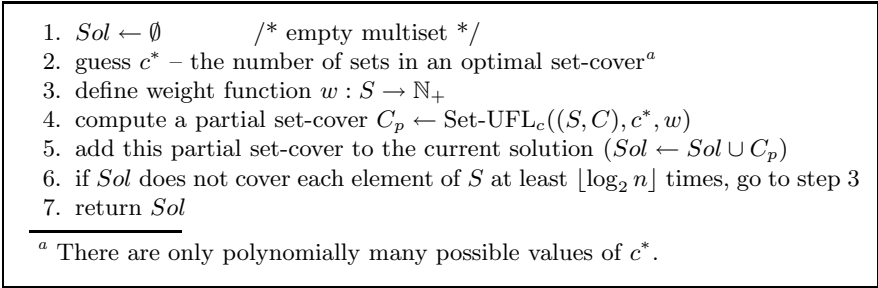


Fig. 1. Algorithm A1

problem for $k = \log_2 n$, where $n = |S|$. A1, given an instance of a $(\log_2 n)$ -SetCover problem (S, C) , produces as a solution a multiset Sol .

Lemma 2. *There exists an (efficiently computable) weight function for step 3 of Algorithm A1, such that the algorithm produces, in polynomial time, a solution to $(\log_2 n)$ -SetCover instance with at most $\mathcal{O}(\log n \cdot c^*)$ sets.*

Proof. We use the Exponential Balancing technique to upper-bound the total number of sets in Sol . Let $c \in (0, 0.768)$ be a fixed parameter of the Algorithm A1. From Corollary 2, in step 4 of Algorithm A1, we may cover a fraction c of weight with at most $\frac{2\lambda_0(1-c)}{3-\lambda_0-2c} \cdot c^*$ sets.

Defining a weight function w in step 3 is like playing a role of player A in the Bucket Game, except for the fact that now the weights are restricted to be positive integers. Fortunately, the proof of Theorem 1 may be easily modified to use only integer, polynomially bounded weights. Hence, by Theorem 1, we may bound the number of used partial covers by $\frac{2\log_2 n}{1-\log_2(2-c)}$.

Concluding, $|Sol|$ - the total number of used sets, may be bounded as

$$|Sol| \leq \frac{4\lambda_0(1-c)}{(1-\log_2(2-c))(3-\lambda_0-2c)} \cdot \log_2 n \cdot c^* . \tag{1}$$

When we set $c = 0.553$ and $\lambda_0 \approx 1.463$, we obtain $|Sol| < 12.006 \cdot \log_2 n \cdot c^*$. \square

Theorem 2. *There exists an algorithm that for any instance of the k -SetCover problem, in polynomial time, produces a k -set-cover with $\mathcal{O}((k + \log n) \cdot c^*)$ sets, where c^* is the number of sets in an optimal classical set-cover.*

Proof. Let $r = \lceil k/\lfloor \log_2 n \rfloor \rceil$, and let $r * Sol$ denote a multi-set containing the same elements as the multi-set Sol , but the quantity of each element e in $r * Sol$ is r times the quantity of e in Sol . We use Algorithm A1 to compute a solution Sol , that covers each element at least $\lfloor \log_2 n \rfloor$ times, and multiply it r times to obtain a solution $r * Sol$. By Lemma 2, Sol has $\mathcal{O}(\log n \cdot c^*)$ sets. Thus, $r * Sol$ has $\mathcal{O}((k + \log n) \cdot c^*)$ sets. \square

Note on Weighted Version of Set Multicover. Like in the case of the classical Set Cover problem, one may generalize the Set Multicover problem

by defining a weight function on the subsets representing the cost of using a particular set in the solution. Same as in the unweighted version, we define the cost of using a set l times to be l times the cost of using this set only once.

All the results concerning the Set Multicover problem (especially Theorem 2) presented in this paper may be easily generalized to the Weighted Set Multicover problem formulation. The proof will be presented in the full version of the paper.

Theorem 3. *There exists an algorithm that for a given instance of the Weighted k -SetCover problem, in polynomial time, produces a k -set-cover with cost at most $\mathcal{O}(k + \log n)$ times the cost of an optimal solution to the classical Weighted Set Cover problem on this instance.*

Motivating Example. Let us consider the following instances of the $(\log n)$ -SetCover problem. Let the family C consist of subsets P_1, P_2, \dots, P_{n+1} with weights c_1, c_2, \dots, c_{n+1} such that $P_j = \{j\}$ and $c_j = 1/j$ for $j = 1, 2, \dots, n$, whereas $P_{n+1} = \{1, 2, \dots, n\}$ and $c_{n+1} = 1 + \epsilon$ for some $\epsilon \in \mathbb{R}_+$. Consider the greedy algorithm of Vazirani, i.e. the algorithm that consecutively chooses the most *cost effective* set. If we use it to produce $(\log n)$ -set-cover for the instance above, it outputs a multi-set containing each of the sets P_1, P_2, \dots, P_n $\log n$ times. This solution has a cost equal to $H_n \cdot \log n$. However, since the optimal solution to the classical Set Cover has cost $c^* = 1 + \epsilon$, by Theorem 3, our algorithm produces $(\log n)$ -set-cover with cost $\mathcal{O}(\log n)$.

4 Application to Dynamic Page Migration

In this section we design a randomized algorithm EBM (Exponential Balancing Marking) and prove that it achieves a competitive ratio of $\mathcal{O}(\sqrt{D \cdot \log n})$. Our algorithm is based on the MARK algorithm [4]. First, we construct a marking scheme, which induces the partition of input sequence into *epochs*. This partition is independent of the algorithm, and depends only on the input. On the basis of the computed marking we construct the EBM algorithm. We also use Bucket Game as an underlying concept, however the relation between EBM and the game is more obscure here.

Marking Scheme. We divide input sequence into chunks of length $K := 2 \cdot \sqrt{D/\log n}$ time steps. The partitioning of input sequence into epochs is performed as follows. Each epoch consists of some non-empty sequence of chunks. Let M_i be the number of marks that v_i has; initially all M_i are set to 0. Marks are DPM's equivalents of the balls from the Bucket Game.

The first epoch starts with the beginning of the input. Let \mathcal{E} denote the current epoch; at the beginning of input sequence $\mathcal{E} = \emptyset$. By a *subsequence* we understand any time interval of the input sequence. For any subsequence \mathcal{S} and any $v_i \in V$, let $A_i(\mathcal{S})$ denote the cost (of serving requests) of an algorithm which remains in v_i for the whole \mathcal{S} and does not move its page. After each chunk I_j we run the marking routine depicted in Fig. 2. Marking of v_i is computed entirely on the basis of $A_i(\mathcal{E})$. If at the end of I_j node v_i becomes marked (with one or

```


$$\mathcal{E} := \mathcal{E} \uplus I_j$$

for each  $v_i \in V$  do  $M_i := \lfloor A_i(\mathcal{E}) \cdot \frac{\log n}{D} \rfloor$ 
if  $M_i \geq \log n$  for all  $v_i \in V$  then
  for each  $v_i \in V$  do set  $M_i := 0$ .
 $\mathcal{E} := \emptyset$  /* beginning of a new epoch */
    
```

Fig. 2. Marking routine after chunk I_j

more marks), i.e. the corresponding value of M_i increases, then I_j is called a *marking chunk* for v_i , and we say that v_i is *marked in* I_j . Additionally, if \mathcal{S} is any subsequence, then by $M_i(\mathcal{S})$ and $M'_i(\mathcal{S})$ we denote the number of marks v_i has before \mathcal{S} and after \mathcal{S} , respectively. We also define $\Delta M_i(\mathcal{S}) = M'_i(\mathcal{S}) - M_i(\mathcal{S})$. An epoch ends when all nodes are marked at least $\log n$ times.

As an important fact, we get that $C_{\text{OPT}}(\mathcal{E}) = \Omega(D)$ for each epoch \mathcal{E} . Actually, if OPT remains at one node v_i , then v_i is marked at least $\log n$ times during \mathcal{E} , and thus OPT pays $A_i(\mathcal{E}) \geq \log n \cdot \frac{D}{\log n} = D$. Otherwise, OPT pays at least D for moving the page between nodes.

Jump Sets. Before we construct our algorithm, we adapt the construction of Jump Sets from [4] for our needs. We consider one single chunk I and we number time steps within I from 1 to K . Then σ_i denotes the node which issues a request in the i -th step of I , and $d_i(\cdot)$, $c_i(\cdot)$ are the distance and cost functions in the i -th step. The following lemma is a simple reformulation of [4-Lemma 2].

Definition 2. A gravity center for I is a vertex v , which minimizes the sum $\sum_{i=1}^K c_K(v, \sigma_i)$. If there is more than one such vertex, then we choose any of them. We denote this node by \mathcal{G}_I .

Definition 3. For any chunk I and any integer $k \geq 1$, a k -JumpSet, which we denote by $J_k(I)$, is the set of all nodes whose distance to \mathcal{G}_I , measured in the last step of I is at most $9 \cdot k \cdot K$, i.e. $J_k(I) = \{v \in V : d_K(v, \mathcal{G}_I) \leq 9 \cdot k \cdot K\}$.

Lemma 3. For any chunk I of K steps, any node $v_i \in V$, and any $k \geq 1$, if $v_i \notin J_k(I)$ at the end of I , then $A_i(I) \geq \frac{k}{4} \cdot K^2 \geq k \cdot \frac{D}{\log n}$. This implies that if v_i gets k marks in I , then $v_i \in J_{k+1}(I)$.

4.1 The EBM Algorithm

Algorithm EBM works in chunks, i.e. it remains at one node for a chunk, and then at the end of the chunk, it makes its decision (where to move the page) on the basis of computed gravity centers and marking. If chunk I is the last chunk of the epoch, then EBM moves its page to \mathcal{G}_I . Otherwise, if the node holding the algorithm's page is marked in I , then at the end of I , EBM chooses randomly a node v^* , further called *I-JumpCandidate*, and moves its page to v^* . Any node v_i is chosen with probability $2^{-M_i} / (\sum_{i \in [n]} 2^{-M_i})$.

Theorem 4. *The algorithm EBM is $\mathcal{O}(\sqrt{D \cdot \log n})$ -competitive.*

Consider any epoch \mathcal{E} , and let m be the number of its chunks, i.e. $\mathcal{E} = (I_1, I_2, \dots, I_m)$. Movements of EBM's page partition \mathcal{E} into a sequence of p phases, i.e. $\mathcal{E} = (P_1, P_2, \dots, P_p)$, each phase consisting of one or more chunks. In one phase P_i , EBM remains at one node, denoted by $P_{\text{EBM}}(P_i)$. First, we prove that the expected number of phases in one epoch is bounded.

Lemma 4. *The expected number of phases in one epoch is $\mathcal{O}(\log n)$.*

Proof. We define a *value of a node* after any chunk I as $n \cdot 2^{-M'_I(I)}$. The *total value* after I is defined as the sum of nodes' values, i.e. $\mathcal{W}_I := \sum_{i \in [n]} n \cdot 2^{-M'_I(I)}$. We make two key observations. First, \mathcal{W}_I is monotonically non-increasing within \mathcal{E} . Second, $\mathcal{W}_I \leq n^2$ for any chunk $I \in \mathcal{E}$, and $\mathcal{W}_{I_{m-1}} \geq 1$ (because after I_{m-1} at least one node has less marks than $\log n$).

EBM starts \mathcal{E} at some node v and remains there till v becomes marked. This first phase lasts for at least one chunk, and after it the total value is at most n^2 . After the first phase, for the analysis, we may safely assume that EBM chooses a jump candidate v^* at the beginning of a phase, moves its page to v^* and remains at v^* till it becomes marked. Thus, this choice of a node v_i determines where the phase ends: either at the first marking chunk for v_i , or at I_m , if v_i is not marked in the remaining part of \mathcal{E} . This chunk we call *stopping* for v_i .

We show that, with probability at least $1/2$, one phase reduces the total value by a constant factor or is the last phase in the epoch. We call such phase *successful*. If the total value is reduced below 1, then the corresponding phase ends with I_{m-1} or with I_m , and thus at most one additional phase consisting of the last chunk I_m is sufficient. Thus, $\mathcal{O}(\log n)$ successful phases are sufficient to finish the whole epoch, and therefore the expected number of phases in epoch is $\mathcal{O}(2 \cdot \log n)$.

Consider the beginning of any phase. We sort the nodes in the order induced by their stopping chunks, obtaining a sorted sequence v_{i_1}, \dots, v_{i_n} . Let p_{i_1}, \dots, p_{i_n} be the probabilities of choosing these nodes as jump candidates. Let j be the smallest index for which $\sum_{k=1}^j p_{i_k} \geq 1/2$, and I' be the stopping chunk for v_{i_j} . Since j is the smallest index with this property, it follows immediately that, with probability $\sum_{k=j}^n p_{i_k} \geq 1/2$, EBM chooses one of $v_{i_j}, v_{i_{j+1}}, \dots, v_{i_n}$ as a jump candidate. Any such choice guarantees that the phase lasts at least to the end of I' . If $I' = I_m$, then the process ends here and the lemma follows. Otherwise, note that between the end of I and the end of I' , $v_{i_1}, v_{i_2}, \dots, v_{i_j}$ are marked at least once. Since probabilities p_{i_k} are directly proportional to the corresponding values of nodes, and $\sum_{k=1}^j p_{i_k} \geq 1/2$, these nodes' values constitute at least half of the total value \mathcal{W}_I . By marking them once, one half of their values (and thus at least $1/4$ of the total value) is removed. Thus, $\mathcal{W}_{I'} \leq \frac{3}{4} \cdot \mathcal{W}_I$. \square

Before we analyze the cost of EBM in a single phase, we introduce the notion of potential Φ . If the distance between nodes holding the pages of OPT and EBM is equal to L , then $\Phi := 2 \cdot D \cdot L$. If \mathcal{S} is any subsequence of steps, then by $\Delta\Phi(\mathcal{S})$ we denote the difference between the potential right after and right before \mathcal{S} .

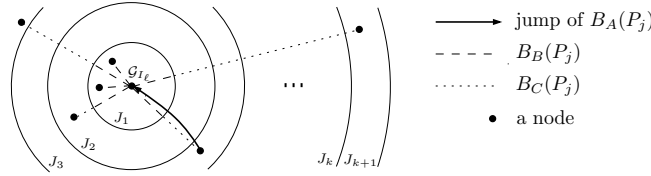


Fig. 3. Transports at the end of phase P_j

By an *amortized cost* of an action we understand the actual cost of this action plus the change in the potential this action induced.

In the following we bound the amortized cost of EBM in any phase P_j . Assume that P_j consists of ℓ chunks numbered from 1 to ℓ , i.e. $P_j = (I_1, I_2, \dots, I_\ell)$, and consider the following thought experiment. At the end of P_j , instead of moving directly to I_ℓ -JumpCandidate v^* , EBM first moves its page to \mathcal{G}_{I_ℓ} , and then to v^* . Note, that for the last phase in \mathcal{E} we do not need the latter part of this movement. Obviously, the (amortized) cost of this combined movement upper-bounds the (amortized) cost of the actual move.

We denote the part of the *amortized cost* EBM pays for serving all the requests in P_j and moving to \mathcal{G}_{I_ℓ} by $B_A(P_j)$. The remaining part of the cost depends on the random choice of v^* . Since we are interested only in the expected value of this variable, we can view this move as transporting parts of the page from \mathcal{G}_{I_ℓ} to the nodes. These transports are schematically presented in Fig. 3. Precisely, to node v_i we transport a part $p_i := 2^{-M'_i(P_j)} / (\sum_k 2^{-M'_k(P_j)})$ of the page, paying $p_i \cdot D \cdot c_K(\mathcal{G}_{I_\ell}, v_i)$. We divide the total *amortized cost* of this transport into two parts: a transport within the boundary of the 1-JumpSet (dashed lines in Fig. 3), denoted by $B_B(P_j)$, and a transport from this boundary to the appropriate nodes (dotted lines in Fig. 3), denoted by $B_C(P_j)$. We note that $B_B(P_j)$ and $B_C(P_j)$ are random variables. The following two lemmas are straightforward generalizations of [5–Lemma 3,4].

Lemma 5. For any phase P holds $B_A(P) \leq \mathcal{O}(D/K) \cdot C_{\text{OPT}}(P) + \mathcal{O}(D \cdot K)$.

Lemma 6. For any phase P holds $B_B(P) \leq \mathcal{O}(D \cdot K)$.

Lemma 7. For any epoch $\mathcal{E} = (P_1, P_2 \dots P_p)$ holds $\mathbf{E}[\sum_{P_j \in \mathcal{E}} B_C(P_j)] = \mathcal{O}(D \cdot K \cdot \log n)$.

The complete proof of Lemma 7 can be found in the full version of the paper. Here we mention only that the proof uses Lemma 3 to argue, that the B_C part of cost in one phase can be high (i.e. the page is transported far away from the gravity center), only if these far nodes received a lot of marks in this phase. This implies that an epoch \mathcal{E} cannot contain many of such phases.

Finally, we can combine the lemmas above to prove EBM’s competitiveness.

Proof (of Theorem 4). Consider any epoch \mathcal{E} and let $\mathcal{E} = (P_1, P_2, \dots P_p)$ be its division into phases. Then, $\mathbf{E}[C_{\text{EBM}}(\mathcal{E}) + \Delta\Phi(\mathcal{E})] \leq \sum_{j=1}^p [B_A(P_j) + B_B(P_j)] +$

$\mathbf{E}[\sum_{j=1}^p B_C(P_j)]$, and by Lemmas 5, 6, and 7, the amortized cost is bounded by $\mathcal{O}(D/K) \cdot C_{\text{OPT}}(\mathcal{E}) + \mathbf{E}[p] \cdot \mathcal{O}(D \cdot K) + \mathcal{O}(D \cdot K \cdot \log n)$. Thus, using $\mathbf{E}[p] = \mathcal{O}(\log n)$ and $C_{\text{OPT}}(\mathcal{E}) \geq D$, we finally get $\mathbf{E}[C_{\text{EBM}}(\mathcal{E}) + \Delta\Phi(\mathcal{E})] \leq \mathcal{O}(\sqrt{D \cdot \log n}) \cdot C_{\text{OPT}}(\mathcal{E})$. By summing this inequality over all the epochs, the proof follows. \square

References

1. Y. Bartal, M. Charikar, and P. Indyk. On page migration and other relaxed task systems. In *Proc. of the 8th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 43–52, 1997.
2. S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms. In *Proc. of the 22nd ACM Symp. on Theory of Computing (STOC)*, pages 379–386, 1990.
3. P. Berman, B. DasGupta, and E. Sontag. Randomized approximation algorithms for set multicover problems with applications to reverse engineering of protein and gene networks. In *Proc. of the 7th Int. Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 39–50, 2004.
4. M. Bienkowski, M. Dynia, and M. Korzeniowski. Improved algorithms for dynamic page migration. In *Proc. of the 22nd Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 365–376, 2005.
5. M. Bienkowski, M. Korzeniowski, and F. Meyer auf der Heide. Fighting against two adversaries: Page migration in dynamic networks. In *Proc. of the 16th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 64–73, 2004.
6. D. L. Black and D. D. Sleator. Competitive algorithms for replication and migration problems. Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University, 1989.
7. S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. In *Proc. of the 9th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 228–248, 1998.
8. D. S. Johnson. Approximation algorithms for combinatorial problems. In *Proc. of the 5th ACM Symp. on Theory of Computing (STOC)*, pages 38–49, 1973.
9. L. Lovász. On the ratio of the optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.
10. M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *Proc. of the 5th Int. Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 229–242, 2002.
11. S. Rajagopalan and V. V. Vazirani. Primal-dual RNC approximation algorithms for set cover and covering integer programs. *SIAM Journal on Computing*, 28(2):525–540, 1999.
12. 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 29th ACM Symp. on Theory of Computing (STOC)*, pages 475–484, 1997.
13. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
14. J. Westbrook. Randomized algorithms for multiprocessor page migration. *DI-MACS Series in Discrete Mathematics and Theoretical Computer Science*, 7: 135–150, 1992.