

A Randomized On–line Algorithm for the k –Server Problem on a Line ^{*}

Béla Csaba^{**1} and Sachin Lodha²

¹ Max-Planck-Institut für Informatik, Saarbrücken, Germany.

² Tata Research Development and Design Center, Pune, India.

{csaba@mpi-sb.mpg.de, sachinl@pune.tcs.co.in}

Abstract. The k –server problem is one of the most important and well studied problems in the area of online computation. Its importance stems from the fact that it models many practical problems like multi-level memory paging encountered in operating systems, weighted caching used in the management of web caches, head motion planning of multi-headed disks, robot motion planning, *etc.* In this paper, we investigate its randomized version for which $\Theta(\log k)$ –competitiveness is conjectured, and yet hardly any $< k$ competitive algorithms are known, even for the simplest of metric spaces of $O(k)$ size.

We present a $O(n^{\frac{2}{3}} \log n)$ –competitive randomized k –server algorithm against an oblivious adversary when the underlying metric space is given by n equally spaced points on a line ($\mathcal{L}(n)$). This algorithm is $< k$ competitive for $n = k + o((\frac{k}{\log k})^{3/2})$. Thus it provides a super–linear bound for n with $o(k)$ –competitiveness for the first time, and improves the best results known so far for the range $n - k \in [o(k), o((\frac{k}{\log k})^{3/2})]$ on $\mathcal{L}(n)$.

1 Introduction

The k –server problem was introduced by Manasse, McGeoch and Sleator in [25]. It is defined as follows: We are given k mobile servers that occupy k points of a metric space \mathcal{M} . We assume that \mathcal{M} is finite, $|\mathcal{M}| = n > k$, and $k > 1$. At each time step a request, a point of \mathcal{M} , appears. An algorithm A must serve this request by moving one of its servers to the requested point (if that is vacant). A is charged for a cost which is equal to the distance moved by the server. The algorithm A is *on–line* if it serves the request without knowing what the future requests will be. An obvious goal is to design an on–line algorithm A that would serve any request sequence ρ most economically.

The standard way of measuring the performance of an on–line algorithm is *competitive analysis* [11, 18]. This measure first appeared in the seminal paper of

^{*} Part of this work was done when both the authors were graduate students at Rutgers University, NJ, USA. The research was also partially supported by DIMACS, see [17].

^{**} Research supported in part by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT), and by OTKA T034475.

Sleator and Tarjan [28] for the paging problem. Its adaptation for the k -server problem gives following definition of competitiveness [25].

Definition 1. *An on-line algorithm A for the k -server problem is c -competitive if for any initial configuration C_0 and any request sequence ρ ,*

$$A(\rho) \leq c \cdot \text{opt}(\rho) + I(C_0),$$

where $A(\rho)$ is the cost incurred by A on ρ , $\text{opt}(\rho)$ denotes the optimal off-line cost of servicing ρ starting at C_0 , and I is a non-negative function depending only on the initial configuration.

Manasse *et al.* [25] proved that the competitive ratio of any deterministic on-line k -server algorithm is at least k . They formulated the famous *k -server conjecture*: There exists a deterministic k -competitive on-line algorithm for every metric space \mathcal{M} . This conjecture is still unsettled, the best upper bound for an arbitrary metric space being $2k - 1$ for WORK FUNCTION algorithm due to Koutsoupias and Papadimitriou [23]. However k -competitive on-line algorithms are known for some special cases. Examples are line [14], tree metric spaces [15], any metric space with $n = k + 1$ [25, 16] or $n = k + 2$ [24, 6], manhattan plane with $k = 3$ [7], *etc.*

Compared to its deterministic counterpart, the randomized k -server problem has yielded very little, especially against *oblivious adversaries* [11, pp. 182]. In the oblivious adversary model [8], there is an adversary who, knowing the randomized algorithm R , but without the knowledge of its random choices, constructs the request sequence ρ , and pays optimally for it (*i.e.* $\text{opt}(\rho)$). In this model, the randomized competitive ratio is defined as follows.

Definition 2. *A randomized on-line algorithm R for the k -server problem is c -competitive against an oblivious adversary if for any initial configuration C_0 and any request sequence ρ ,*

$$E[R(\rho)] \leq c \cdot \text{opt}(\rho) + I(C_0), \tag{1}$$

where $E[R(\rho)]$ denotes the expected cost incurred by R on ρ , $\text{opt}(\rho)$ denotes the optimal off-line cost of servicing ρ starting at C_0 , and I is a non-negative function depending only on the initial configuration.

Less famous, but equally frustrating, *randomized k -server conjecture* states that there is a $O(\log k)$ -competitive randomized k -server algorithm against an oblivious adversary for every metric space. Yet after substantial efforts, only few results are known, and the best ones are possible for some special metric spaces alone. For the uniform metric space, matching upper and lower bounds of $H_k = 1 + \frac{1}{2} + \dots + \frac{1}{k} \approx \log k$ are known. McGeoch and Sleator [26] presented the upper bound with highly sophisticated PARTITION algorithm, and Fiat *et al.* [19] provided the lower bound. For the case of 2 servers on an isosceles triangle, Karlin *et al.* [22] showed matching upper and lower bounds of $\frac{e}{e-1}$. Blum *et*

al. [9] studied (among others) the case of $p(n)$ -unbalanced metric spaces, and proved $\Theta(\log k)$ competitiveness for the same. Recently, using unfair metrical task systems technique, Seiden [27] has presented a $O(\text{polylog}(k))$ -competitive randomized algorithm for metric spaces which can be separated into a small number of widely separated sub-spaces.

Otherwise situation is not very pleasant, and we do not even know randomized algorithms with competitive ratio $< k$ for very simple metric spaces, *e.g.* line! All the best known results have been obtained as corollaries of results for the metric spaces ([1, 2]) and the metrical task system (MTS) problem ([12, 1, 3, 9, 20]). Translating these results for the k -server problem, the smallest randomized competitive ratio for general metric spaces is $\min(2k - 1, O(\log \binom{n}{k} \cdot \text{polylog}(n)))$ due to Fiat and Mendel [20]. Note that it has a large divide with the best known lower bound of $\Omega(\frac{\log k}{\log^2 \log k})$ given by Bartal, Bollobás and Mendel in [4], and even for $n = O(k)$, this competitive ratio is no better than $O(k)$.

In this paper we investigate the randomized k -server problem for the special case when the underlying metric space is given by n equally spaced points on a line, the metric space we denote by $\mathcal{L}(n)$. A line (or $\mathcal{L}(n)$) as the metric space has received special attention in literature for its simplicity and importance in robot motion planning problem [10, 9], motion planning of 2-headed disks [13], *etc.* In fact, Blum *et al.* [9] prove the lower bound of $\frac{\log k}{\log \log k}$ on the randomized competitive ratio for $\mathcal{L}(n)$ which is pretty close to conjectured $\log k$ bound, and yet hardly any $< k$ competitive randomized algorithms are known for line (or $\mathcal{L}(n)$) even after restricting k . The best known results are as follows: For the case when the underlying metric space is the real line and $k = 2$, Bartal *et al.* [5] present an algorithm with competitive ratio $\frac{155}{78} < 2$. Next natural case of 3 servers on real line remains challenging open problem. For $\mathcal{L}(n)$, there is a randomized strategy due to Fiat and Mendel [20] based on their MTS result. It has competitive ratio $O(\log \binom{n}{k} \log n)$ which is $< k$ competitive only if $n = k + o(k)$.

Our contribution is a $O(n^{\frac{2}{3}} \log n)$ -competitive randomized on-line algorithm \mathcal{R} for $\mathcal{L}(n)$. In fact, for any request sequence ρ , we show that the expected cost of \mathcal{R} ,

$$E[\mathcal{R}(\rho)] < O(n^{\frac{2}{3}} \log n) \cdot \text{opt}(\rho) + o(n^2).$$

Note that the additive term $o(n^2)$, denoting the function I in the equation (1), depends on the size of the metric space, *i.e.* n , itself. Since n is finite, it is a constant independent of the initial configuration.

\mathcal{R} is $< k$ competitive for $n = k + o((\frac{k}{\log k})^{3/2})$. Thus it provides a super-linear bound for n with $o(k)$ -competitiveness for the first time, and improves the best results known so far for the range $n - k \in [o(k), o((\frac{k}{\log k})^{3/2})]$ on $\mathcal{L}(n)$. It also works well for few other special metric spaces, improving results therein. Examples are circle with equally spaced n points, ladder, *etc.* We remark that our method is directly suited for the k -server problem, we do not use previous results of the MTS.

2 Outline

We sketch a brief outline of the paper here. The evader problem and oblivious adversary model are described in section 3, for we will define and analyze our algorithm as a randomized evader algorithm. In section 4, we introduce basic ideas of our algorithm through the simplest case $n = k + 1$. It serves as a launchpad for section 5 which details the main algorithm \mathcal{R} . We describe the notation for remainder of the paper in section 6. Sections 7 and 8 are devoted to analysis of \mathcal{R} where we find upper bound for expected cost of \mathcal{R} in terms of jumping evaders. We analyze the lower bounds for optimal cost in section 9 and relate them to the expected cost of \mathcal{R} in section 10, finally proving our competitiveness result.

3 Model

Throughout the paper, we consider the k -server problem in an equivalent form of l -evader problem [24]: There are $l = n - k$ evaders on the metric space \mathcal{M} with n points. When there is no server on a point $x \in M$, then there is an evader on x , and vice versa. While in the k -server problem the satisfaction of a request is moving a server there, in the l -evader problem we must move away the evader residing on the request. Off-line and on-line algorithms, competitiveness are defined in the same way as in the k -server setup. Note that no two evaders can occupy the same point of \mathcal{M} since that would result in more servers. It is easy to see that the l -evader problem is an equivalent formulation of the k -server problem. We will define and analyze our algorithm as a randomized l -evader algorithm. We will also like to remind the reader that we are working exclusively against an *oblivious adversary* [8].

4 Warm-up

In this section we illustrate the underlying ideas of the algorithm \mathcal{R} in its simplest form. In fact, we present a randomized algorithm \mathcal{Q} , forerunner of \mathcal{R} , for the special case $k = n - 1$, *i.e.* the case of only 1 evader. We denote this evader by symbol q . Please note that this description of \mathcal{Q} is not very formal or precise, its purpose is to develop insight and facilitate understanding of \mathcal{R} .

The idea of \mathcal{Q} (or \mathcal{R}) is to break the $\mathcal{L}(n)$ into a collection of equal-size line segments or blocks, and keep the evader q in a block until the block becomes *saturated* with requests. Then \mathcal{Q} *marks* the saturated block, randomly chooses a new unsaturated block and moves q to this block, and so on. This idea is similar to the one used by Borodin, Linial and Saks [12] in their randomized MTS algorithm. It also appears in the CIRC2 algorithm [11, pp. 184-5] which is an adaptation of a line algorithm due to Blum *et al.* [10].

Incidentally CIRC2 algorithm is designed for the 1-evader problem on a circle having equally spaced n points. It differs from \mathcal{Q} in its strategy of servicing requests inside a block. CIRC2 uses any k -competitive deterministic algorithm,

such as DOUBLE-COVERAGE [14], inside a block till the block gets saturated, whereas \mathcal{Q} uses a very naive and simple strategy for the same. The two also differ in their block partitioning strategy, CIRC2 does it deterministically whereas \mathcal{Q} introduces randomness in it. In final analysis, both the algorithms are $O(\sqrt{k} \log k)$ -competitive.

4.1 Algorithm \mathcal{Q}

We develop algorithm \mathcal{Q} now. We start by partitioning the line segment $\mathcal{L}(n)$ of length n into $t = \sqrt{n}^1$ contiguous blocks, namely B_1, B_2, \dots, B_t , each having $b = n/t = \sqrt{n}$ points. Thus the block B_i contains the points $\{(i-1)\sqrt{n}, (i-1)\sqrt{n}+1, \dots, i\sqrt{n}-1\}$, where $1 \leq i \leq t$. Without loss of generality, we may assume that two consecutive points of $\mathcal{L}(n)$ are unit distance apart. So each block has length $b = \sqrt{n}$.

The possible motions of q can be either taking steps inside a block, or jumping from a block into another one. Let us describe how q moves inside a block B . In the beginning, say, q is at the leftmost position of the block, and its starting *direction* is “right”. If a request comes which forces q to move², then it takes one step to the “right”, if its current position is not the rightmost point of B . Otherwise q changes its direction from “right” to “left”, and takes one step to the left. After several requests, if q reaches the other endpoint, the leftmost position of B , then it changes direction again. In general, q changes direction whenever it reaches an endpoint of B . We say that q moved a *round* if it switches its direction from “left” to “right”. If q completes \sqrt{n} rounds in block B , that is, $O(n)$ steps, then \mathcal{Q} *marks* B (as saturated) and moves q to another *unmarked available* block.

Let’s consider the situation where the only evader a of the adversary \mathcal{A} and q both are residing in the same block B . Then during each round of q in B , a would need to move at least twice inside B ! Hence, by the time \mathcal{Q} marks B , a must have moved $\Omega(\sqrt{n})$ times. This, in nutshell, implies $O(\sqrt{n})$ -competitiveness.

This would work fine if \mathcal{Q} knew a ’s current position! But, being online, \mathcal{Q} doesn’t know a ’s whereabouts. Therefore it needs to explore the entire block structure. Rather than keeping q in the same block forever, \mathcal{Q} moves q to a randomly chosen unmarked block, but only after q has done enough work in B to justify this jump. *Ski Principle* [21, pp. 522] comes into picture here: To account for jump out cost of $O(n)$ (since two blocks can be $O(n)$ away), q does $O(n)$ work in a block, \mathcal{Q} *marks* that block (as saturated), and then q jumps out into an unmarked block.

If q moves through each and every block, then we are once again in trouble. We end up paying $\Theta(n\sqrt{n})$, while \mathcal{A} pays only $\Omega(\sqrt{n})$, that is, $O(n)$ -competitiveness in worst case. In fact, while q is exploring a particular block

¹ Without loss of generality, and for simplicity of notation and discussion, we assume that \sqrt{n} is a whole number.

² \mathcal{Q} is a *lazy* algorithm. If the request is not for the current position of q , then it makes no moves.

B , lots of requests are being placed in other blocks as well. We would want to take advantage of this and reduce the number of blocks being explored by q . How do we achieve this? We do following: In order to keep track of requests made in B_i , we associate one *auxiliary block*, say A_i , with it. Like B_i , A_i has b points. An auxiliary block is used for book-keeping purpose alone and is not part of the $\mathcal{L}(n)$. A_i contains one evader and rest all servers, that is, image of B_i containing q . Whenever a request is made to a point in B_i , we satisfy it in B_i as well as in A_i . Thus we can keep account of requests made and amount of work being done in B_i irrespective of whether q is in B_i or not. And we use the work done in A_i as an indicator for saturation of B_i ! That is, \mathcal{Q} marks B_i if \sqrt{n} rounds are completed in A_i . Notice that if a were in B_i , then $\Omega(\sqrt{n})$ bound on \mathcal{A} 's cost remains true, irrespective of q itself being in B_i or not.

Naturally \mathcal{Q} works in phases. At the start, all blocks are unmarked, and therefore available. During satisfaction of a request sequence, blocks get marked. q must always be in an unmarked block. So whenever its current block gets marked, it has to choose an unmarked block randomly and uniformly, and jump into it. When all the blocks are marked and q has no block to jump into, \mathcal{Q} erases all the marks, resets cost counter for each block to zero, and starts a new phase. Note that, for a given request sequence, phases start and end independently of \mathcal{Q} 's random choices. This is true since marking depends only on the request sequence. Hence one can calculate the expected number of jump-outs of q in a phase. Intuitively the number of unmarked blocks get *halved* between consecutive jump-outs of q . Therefore the expected number of jump-outs of q in a phase should be $O(\log t) \ll t$. Rigorous analysis shows that this number is, in fact, the t^{th} harmonic number $H_t = \frac{1}{2} \log n + O(1)$. Therefore the expected cost of \mathcal{Q} in a phase is $O(n \log n)$.

What is the cost of \mathcal{A} in a phase? If we assume that a remains in one block for the whole phase, then it must move $\Omega(\sqrt{n})$ steps. But this assumption is artificial. In fact a can change blocks too, and the cost of \mathcal{A} can be as small as 1 in a phase: Consider case when a is at the leftmost position of B_2 at the start of a phase. \mathcal{A} gives a request sequence which forces \mathcal{Q} mark B_1 right away. Then a moves only 1 step to its left and sits in B_1 right till the end of the phase!

This strategy of \mathcal{A} is based on its knowledge of block boundaries. Therefore we hide this information from \mathcal{A} . Before any request appears, \mathcal{Q} picks a random *shift* s uniformly from the set $\{0, 1, 2, \dots, \sqrt{n}-1\}$ and shifts the block system, *i.e.* the i^{th} block will contain the points $\{(i-1)\sqrt{n}+s, (i-1)\sqrt{n}+s+1, \dots, i\sqrt{n}+s-1\}$ for $2 \leq i \leq t-2$. Points $\{0, 1, \dots, \sqrt{n}+s-1\}$ constitute the 1st block while points $\{(t-2)\sqrt{n}+s, (t-2)\sqrt{n}+s+1, \dots, \sqrt{n}\}$ constitute the final $(t-1)^{\text{th}}$ block. Thus, all the blocks are of length $O(\sqrt{n})$.

After this initial shift, \mathcal{Q} acts the same way as described above. It moves $O(n \log n)$ steps on average in a phase. On the other hand, we expect \mathcal{A} to work $\Omega(\sqrt{n})$ during a phase owing to its ignorance about shift s . Careful computation shows that \mathcal{Q} is indeed a $O(\sqrt{n} \log n)$ -competitive randomized 1-evader algorithm on $\mathcal{L}(n)$. In other words, \mathcal{Q} is $O(\sqrt{k} \log k)$ -competitive k -server algorithm on $\mathcal{L}(k+1)$.

Remark: Note that \mathcal{Q} is outperformed by many others in the literature. The original line version of CIRC2 [10] is $2^{O(\sqrt{\log k \cdot \log \log k})}$ -competitive. Fiat and Mendel [20] present $O(\log^2 k)$ -competitive algorithm for the same case. Though inferior, \mathcal{Q} provides foundation for our main algorithm \mathcal{R} described in the next section. \mathcal{R} is $o(k)$ -competitive even when $n \gg k$.

5 Algorithm \mathcal{R}

5.1 Partitioning $\mathcal{L}(n)$

We start by determining a block system on the line segment. First of all, we *partition* it into $\frac{n}{b}$ blocks, each of length $b = n^{\frac{1}{3}}$.³ Secondly, we pick a random number s uniformly from the interval $[0, b-1]$ and shift the blocks by this number. We glue together the first two and the last two blocks so that the length of the leftmost block is $b + s$, the length of the rightmost block is $2b - s$, all the other blocks have the same length b . Thus total number of blocks is $t \approx \frac{n}{b}$. We denote the length of block B by notation $|B|$. Note that $b \leq |B| < 2b$ always.

5.2 Evader Motion Rules

During the satisfaction of a request sequence ρ , the evaders of \mathcal{R} will have two type of motions: steps inside a block or randomly jumping out into one of the *available* blocks. The rules for what makes a block available will be discussed later.

Let us provide the rules for taking steps inside a block. Assume that there are r evaders in block B . We will consider first the case $r < |B|$, the other case, $r = |B|$, will be handled differently. Let ρ be a request sequence. If the i^{th} request of ρ , ρ_i , is not in B , then no evader will move in B . Otherwise, if $\rho_i \in B$ and there is an evader on ρ_i , that evader will go to the closest vacant position according to its *direction*. Every evader has a direction. It is “right” in the beginning, and from time to time, it changes from “right” to “left” and vice versa. Suppose that evader e is residing on ρ_i and its direction is “right”. Then e will move to the closest vacant position to the right of ρ_i . If there is no such position, e changes its direction to “left” and will go to the closest vacant position to the left of ρ_i . Similarly, e switches direction from “left” to “right”, if it has to move, and there is no vacant position on the left. These are the only cases an \mathcal{R} -evader changes its direction. We say that e moved a *half-round* whenever it switches its direction. We say that e moved a *round* when it switches its direction from “left” to “right”. Note that we have ensured that no two evaders will take the same position in any block.

When $r = |B|$ and there is a request for a position in B , that evader residing on the request will jump out into an available block.

³ Without loss of generality, and for simplicity of notation and discussion, we assume that $n^{\frac{1}{3}}$ is a whole number.

5.3 Auxiliary Blocks and Marking Procedure

\mathcal{R} divides the course of satisfaction into phases. At the start of a phase, all blocks are available. The phase ends when all blocks become *unavailable*. For deciding which are the available blocks and how to change blocks for evaders, we assign $|B|$ *auxiliary blocks* $AB_1, \dots, AB_{|B|}$ to each block B . Each of them has length $|B|$. As in case of \mathcal{Q} , auxiliary blocks are used for book-keeping purpose alone and are not part of the $\mathcal{L}(n)$. Also note that \mathcal{R} does not pay for work done in any of the auxiliary blocks. During a phase, every block gets *marks* based on working of *auxiliary evaders* moving in corresponding auxiliary blocks. These marks not only help \mathcal{R} to decide on the availability of a block where an evader can jump-out to, but also lower bound the cost of an adversary as we shall see in Lemma 9.

Right at the start of the algorithm, \mathcal{R} puts x evaders into the auxiliary block AB_x for $1 \leq x \leq |B|$. The evaders in AB_x are placed on the leftmost x positions in AB_x and their starting direction is “right”. These auxiliary evaders move in AB_x satisfying ρ in the same manner as we described above for the \mathcal{R} -evaders. Of course, the auxiliary evaders don’t jump out anywhere! We count the number of rounds taken by *all* the auxiliary evaders of AB_x from the beginning of the phase. If the sum of these rounds in AB_x reaches $8b^2$, then the block B is x -marked. For technical purpose, block B is $|B|$ -marked immediately when the first request for a point inside it appears during the phase. Note that a block can have several marks. We always consider the *smallest* one, and denote it as $m(B)$. Thus, for each block, its mark, $m(B)$, is monotonically decreasing with time during a phase. Observe that the marking procedure is *deterministic*, there is *no* random choice in it.

5.4 Description of \mathcal{R}

Initialization: \mathcal{R} chooses a random shift $s \in [0, b - 1]$ and determines the block system as described in subsection 5.1. \mathcal{R} places its evaders on the leftmost positions inside their blocks. The auxiliary evaders also take the leftmost positions in the auxiliary blocks.

Description of a Phase:

- If there is a request for a position in block B , take steps in B and in AB_j ($1 \leq j \leq |B|$) according to the rules described in subsection 5.2 and update B ’s mark, $m(B)$, if necessary (as mentioned in subsection 5.3).
- If there are $r \geq m(B)$ evaders in B , then select $r - m(B) + 1$ evaders for ejection from B — *if possible*, choose among those which already changed block during this actual phase, and order them using *First In First Out* policy.⁴

⁴ These are only technical matter that are needed for cost accounting later. Relabelling of evaders can easily achieve these goals.

- Determine the available blocks: A block having r' \mathcal{R} -evaders is available if its mark $m(\cdot) > r' + 1$. Pick randomly, uniformly a block C from the available ones. Let's suppose it has r'' evaders. The first of the to-be-ejected evaders jumps into it. Rearrange the evaders of the block C to have the configuration of the evaders of the auxiliary block $AC_{r''+1}$. Rearrange the evaders in block B according to the configuration of the evaders of the auxiliary block AB_{r-1} . For each of the remaining to-be-ejected evaders, repeat the above process: Randomly pick an available block for jumping in, the evader jumps into it, then rearrange the evaders in this new block and those in block B according to the actual configuration of the evaders in the appropriate auxiliary blocks.
- If there is no available block for a to-be-ejected evader, the phase ends. Erase marks of all the blocks and begin a new phase.

We finish the definition of \mathcal{R} . As we already stated, \mathcal{R} starts by determining the block system on the line segment before the first request appears. The shift s is *not known* to \mathcal{A} . The evaders are placed according to the initial configuration. \mathcal{R} rearranges them to their leftmost positions inside each block. Then the first phase starts when the first request appears. Please note that there is *no* rearrangement of \mathcal{R} -evaders (to any specific configuration) inside the blocks at the start of any subsequent phase.

Remark: Note that the marking procedure has been designed to ensure following: Whenever some evader jumps out of block B at some time in a phase, then B becomes *unavailable* for the rest of that phase, *i.e.* there will be no evader jumping in B during the rest of the phase. Essentially evader jump-outs from block B imply it being already *saturated* with requests. So we do not send any more evaders there! This fact plays very crucial role in the analysis of \mathcal{R} .

6 Notation

We use following notation throughout the remainder of the paper.

- We assume that our blocks are arbitrarily numbered from 1 to t . So we can talk about i^{th} block, B_i .
- C_p denotes the *block-wise configuration* of the adversary evaders at the end of Phase p , *i.e.* C_p is a t -tuple: It has the number r in the i^{th} position, $C_p(i) = r$, if at the end of Phase p there are r \mathcal{A} -evaders in B_i .
- Similarly D_p denotes the block-wise configuration of the \mathcal{R} -evaders at the end of Phase p .

7 Analysis of \mathcal{R}

We are going to prove the following theorem.

Theorem 1 For any initial configuration and for any request sequence ρ ,

$$E[\mathcal{R}(\rho)] < O(n^{\frac{2}{3}} \log n) \cdot \text{opt}(\rho) + o(n^2).$$

Thus \mathcal{R} is a $O(n^{\frac{2}{3}} \log n)$ -competitive randomized k -server algorithm on $\mathcal{L}(n)$. It is $o(k)$ -competitive for $n = k + o((\frac{k}{\log k})^{3/2})$.

7.1 \mathcal{R} Is Distribution Invariant

Once we fix the shift s of the block system, we can think of \mathcal{R} as an algorithm that moves from the current arrangement of the l evaders on $\mathcal{L}(n)$ to another one in response to the request ρ_i depending on the outcome of coin-tosses as well as the current marks of the blocks. Given c_0 as the initial arrangement of the evaders on $\mathcal{L}(n)$, a sequence $(c_0, c_1, c_2, \dots, c_{|\rho|})$ of arrangements is called a ρ -satisfying run (or sometimes only run) of \mathcal{R} if there exist coin-tosses that make \mathcal{R} move from arrangement c_{i-1} to c_i in response to the request ρ_i for $1 \leq i \leq |\rho|$. Thus there can be many ρ -satisfying runs due to \mathcal{R} 's random choices. We show that, despite this apparent randomness, the distribution of \mathcal{R} 's evaders is the same for all runs at the end of each phase for a fixed shift.

Lemma 1 Fix the shift s of the block system and let ρ be an arbitrary request sequence. Let B be a block. Suppose that for one run of \mathcal{R} there are r evaders in B at the end of Phase p when satisfying ρ . Then,

- B contains exactly r evaders at the end of Phase p for all ρ -satisfying runs of \mathcal{R} , and
- each and every phase ends at the same time (i.e. after the same request) for all runs.

Proof: We prove the statement by induction on the phase number. It is trivially true before the first request appears because \mathcal{R} starts in the initial configuration in each of its runs. Assume that the lemma is true up to Phase $p - 1$. Assume that we are now at the end of Phase p for the run f_1 and B has r evaders inside. We also assume that f_1 is the first run which finished Phase p , i.e. other runs may not have finished the Phase p so far.

We will denote the (smallest) mark of block j by $m(j)$. $e_x(j)$ will track the current number of evaders in block j in run x . Recall that $m(j)$ is invariant of random choices of algorithm, i.e. different runs, for a given request sequence ρ . The current available space in block j in run x is $s_x(j) = m(j) - e_x(j) - 1$. Note that $s_x(j) \geq 0$ for any x, j always. We end phase when we cannot fulfill above requirement for every block j .

Since f_1 is the first run to end Phase p , there must be some block i_0 such that its latest mark $m(i_0)$ caused $e_1(i_0) - m(i_0) + 1$ evaders to jump out of block i_0 and there is no space left for them. Mathematically, the available space

$$S_1 = \sum_{j, j \neq i_0}^t s_1(j) < e_1(i_0) - m(i_0) + 1.$$

Note that S_1 can also be written as

$$\begin{aligned}
S_1 &= \sum_{j,j \neq i_0}^t m(j) - \sum_{j,j \neq i_0}^t (e_1(j) + 1) \\
&= \sum_{j,j \neq i_0}^t m(j) - (l - e_1(i_0) + t - 1) \\
&= \left(\sum_{j,j \neq i_0}^t m(j) - l - t + 1 \right) + e_1(i_0), \tag{2}
\end{aligned}$$

where the first term in equation (2) is same for different runs owing to deterministic marking procedure.

Let's examine situation in run f_2 when jump-outs start in block B_{i_0} in run f_1 . We consider three cases:

1. $e_1(i_0) = e_2(i_0)$: In both the runs, evaders start jumping out of block, one by one, simultaneously. Suppose we have problem for w^{th} evader jumping out in the run f_1 . At this point in time, $S_1 = 0$. That means $e_1(j) = m(j) - 1$ for all j ($j \neq i_0$). Corresponding sum S_2 is also 0, owing to the fact that marks are same and the current number of evaders in block i_0 is same for both f_1 and f_2 . So f_2 also ends and for any block j , $e_1(j) = e_2(j)$.
2. $e_1(i_0) > e_2(i_0)$: Here we have to distinguish two cases
 - (a) $e_1(i_0) \geq m(i_0) > e_2(i_0)$: There are no jump-outs in f_2 , and yet S_1 becoming 0 after w^{th} jump-out in f_1 indicates that S_2 was already negative to begin with — a contradiction!
 - (b) $e_2(i_0) \geq m(i_0)$: There will be jump-outs in f_2 now. We can envision this as f_2 waiting for f_1 to bring its $e_1(i_0)$ value to $e_2(i_0)$ and then starting its own jump-outs. If, at all, $e_1(i_0)$ could be reduced to $e_2(i_0)$, then we are in case 1. If not, then we have $S_2 < S_1 = 0$ — a contradiction!
3. $e_1(i_0) < e_2(i_0)$: There are jump-outs in both f_1 and f_2 . We can envision this as f_1 now waiting for f_2 to bring its $e_2(i_0)$ value to $e_1(i_0)$ and then starting its own jump-outs. Again we are in case 1!

This finishes the proof of the lemma. □

Remark: This is a very important lemma which allows us to calculate the exact number of evaders that changed blocks during any Phase p . This number is

$$m_p = \sum_{i=1}^t \max\{0, D_{p-1}(i) - D_p(i)\}.$$

Since $l = \sum_{i=1}^t D_{p-1}(i) = \sum_{i=1}^t D_p(i)$, we can also write following expression for m_p

$$m_p = \frac{1}{2} \sum_{i=1}^t |D_{p-1}(i) - D_p(i)|. \tag{3}$$

Note that every block and its corresponding all auxiliary blocks have exactly the same configuration at the end of a phase irrespective of random choices made during the phase. Once we fix s and ρ , we fix D_p 's. We will like to figure out how *quickly* we move from D_{p-1} to D_p in \mathcal{R} . We answer it in the next section.

8 Expected Cost of \mathcal{R}

We now prove a series of lemmas to estimate the expected cost of \mathcal{R} .

8.1 Initial Cost

Lemma 2 \mathcal{R} spends at most $O(bn)$ during the **Initialization**.

Proof: \mathcal{R} rearranges all the evaders in each and every block during the **Initialization**. This could be done in $O(b^2)$ cost per block. Summing over all blocks, cost paid by \mathcal{R} is at most $O(b^2) \cdot t = O(bn)$. \square

Note that this is a one-time cost and it can be absorbed in the constant function I of equation (1).

8.2 Cost Associated with Jump-outs

Let us now consider a simple game related to the analysis of \mathcal{R} . Suppose the adversary has a permutation σ of the first t positive integers $[t]$. $\sigma(i)$ denotes i^{th} element of σ and, therefore, $\sigma^{-1}(i)$ denotes the position of i in σ . We *don't know* σ . In the first round, we randomly, uniformly pick one element of $[t]$: t_1 . Let $p_1 = \sigma^{-1}(t_1)$. Then adversary shows us first p_1 elements of σ and discards them. In the second round, we randomly, uniformly pick one element from the remaining ones, denote it by t_2 . In general, in the $(i+1)^{\text{th}}$ round, we pick a number randomly, uniformly from the set $\{\sigma(p_i+1), \sigma(p_i+2), \dots, \sigma(t)\}$ and denote it by t_{i+1} . We stop whenever we run out of elements, *i.e.* reach $\sigma(t)$. Let's denote the number of rounds in one possible outcome of the game by l_t — this is a random variable. Let's denote its expectation, $E[l_t]$, by L_t .

Lemma 3 $L_t = \sum_{i=1}^t \frac{1}{i}$. That is, $L_t = \log t + O(1)$.

Proof: We prove the statement by induction on t . It is trivially true for $t = 1$. Assume now that the lemma is true up to t . Consider the set $\{1, 2, \dots, t, t+1\}$. Pick t_1 and let $p_1 = \sigma^{-1}(t_1)$. p_1 can be anything between 1 and $t+1$ with the same probability $\frac{1}{t+1}$. From this, we can write recurrence: $L_{t+1} = \frac{1}{t+1} \sum_{r=1}^{t+1} (1 + L_{t+1-r})$; ($L_0 = 0$). Using the induction hypothesis and simple rearrangement, we get

$$L_{t+1} = L_t + \frac{1}{t+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t+1} = \sum_{i=1}^{t+1} \frac{1}{i}.$$

\square

Remark: Note that Lemma 3 is valid even if we allow adversary following freedom: After we pick an element in any round, adversary can pick some permutation of the remaining elements from *any* probability space defined on the set of possible permutations of those remaining elements for the next round. This is clearly due to our unawareness of both: the σ itself and adversary's choice of probability space.

Recall that when an \mathcal{R} -evader has to jump out of a block, that evader jumps out which is among the ones that jumped into the block in the actual phase, if there is such an evader. This allows us to talk about the number of evaders changing block during a phase. We will like to relate the *number of jumping evaders* and the *expected number of jump-outs* during a phase.

Let's consider the j^{th} request, say ρ_j . Suppose it is in block B_i . When \mathcal{R} serves it, that might possibly change mark of B_i depending on work done in B_i 's auxiliary blocks. So if we keep track of these changes in the marks of the blocks while serving ρ , we get a *unique* marking sequence corresponding to ρ . Even though the marking sequence is deterministic, randomization comes into play when we choose arbitrary available block for a jumping evader. So any marking sequence will have lots of possible runs. From Lemma 1, we know that all these runs start and end all phases at the same time and have exactly same distribution of evaders at the end of each and every phase. This allows us to consider restriction of the marking sequence, and therefore associated runs, to any one particular Phase p . From the remark at the end of Lemma 1, we know that total number of evaders that change block during Phase p is

$$m_p = \frac{1}{2} \sum_{i=1}^t |D_{p-1}(i) - D_p(i)|.$$

Let M_p be a random variable which counts the number of jump-outs during Phase p . M_p may take different values depending on the run. We will prove following lemma that finds the expectation of M_p .

Lemma 4 *Suppose there are m_p evaders which change block during Phase p . All the others stay in their respective blocks for the whole phase. Then the expected number of jump-outs in Phase p , $E[M_p]$, is $O(m_p \log n)$.*

Proof: There are m_p jumping evaders, let's use random variable Y_i to count the number of jump-outs of i^{th} jumping evader in Phase p . Then, $M_p = \sum_{i=1}^{m_p} Y_i$.

We will first find out $E[Y_i]$. Let's consider the original block, say B_j , from which our i^{th} evader, e_i , starts jumping out. There are precisely $b_j = D_{p-1}(j) - D_p(j)$ evaders that *start* their jump-outs from B_j . Let e_i be w^{th} evader, $1 \leq w \leq b_j$, that jumps out from B_j . Then it is clear that, irrespective of the run, e_i will start jumping out only when the mark of B_j drops below $D_p(j) + w + 1$ for the *first time*. Let's denote the time of this event in the marking sequence by T_i .

At T_i , depending on the run we had *so far*, there is a certain configuration of evaders in different blocks. Let's suppose that $x(< t)$ blocks are available for e_i to jump into, namely, $B_{i_1}, B_{i_2}, \dots, B_{i_x}$, and the block B_{i_q} has r_{i_q} evaders, where $1 \leq q \leq x$.

Recall that we use **FIFO** rule for ejecting evaders from any block: An evader will not jump out of a block unless those which jumped into the same block *after* it have already jumped out.

This will mean that if e_i chooses a block B_{i_q} to jump into, $1 \leq q \leq x$, then it will jump out only when mark of the block B_{i_q} drops below $r_{i_q} + 2$ for the *first time*. Let's denote the time of this event in the marking sequence by T_{i_q} .

It is easy to see that we are in the situation described for the game of Lemma 3. The *original* σ in Lemma 3 corresponds to the permutation of i_1, i_2, \dots, i_x that sorts $T_{i_1}, T_{i_2}, \dots, T_{i_x}$ in the *increasing* order. This is the order in which blocks become *unavailable* irrespective of whatever random choices made by other jumping evaders. The evader e_i has no idea about σ and picks any one of the x available blocks. On the other hand, adversary has no inkling of e_i 's random choices. Therefore the expected number of jump-outs of e_i for the set of possible runs hereafter is at most $\log x + O(1) = O(\log t)$. Averaging over set of all initial runs (till the start of e_i jump-out), $E[Y_i] = O(\log t)$.

Please note that e_i might not have to jump out even if adversary *hits* its block. Secondly, some blocks might become unavailable even before their *time comes* as indicated by σ ! Both these observations strengthen our claim made above. Now it trivially follows that

$$E[M_p] = \sum_{i=1}^{m_p} E[Y_i] = O(m_p \log t) = O(m_p \log n).$$

□

Lemma 5 *If there are m_p evaders during Phase p which change block, then the expected cost of \mathcal{R} associated with jump-outs during Phase p is $O(m_p n \log n)$.*

Proof: Let's fix a run and say there are M_p jump outs during Phase p . We can associate following costs to a jump out:

1. the work done inside the block before a jump-out — it is $O(n)$. Reason: Jump-out from block B takes place when its mark, $m(B)$, becomes less than or equal to the current number of evaders, say r , in it. This means that auxiliary block $AB_{m(B)}$ finished $O(b^3) = O(n)$ work *since start of the Phase p* . Since we keep the smallest mark, work done in AB_r (which is exact replica of B !) *since start of the Phase p* is at most $O(n)$.
2. the distance the evader jumps — it is $O(n)$; and
3. the cost of rearranging the blocks — it is $O(b^2) = o(n)$.

So overall cost is $O(n)$ per jump-out and the cost associated with jump-outs during Phase p is $O(M_p n)$.

But notice that M_p is a random variable and, from Lemma 4, $E[M_p] = O(m_p \log n)$. Therefore, the expected cost of \mathcal{R} associated with jump-outs during Phase p is $O(m_p n \log n)$. □

8.3 Overhead Cost

Mere jump-outs don't account for entire cost paid by \mathcal{R} . We have to account for

1. work done in a block after final jump-out out of it,
2. work done in a block after final jump-in into it, and
3. work done in a block with no jump-ins and jump-outs.

Each type of work is of order $O(n)$. Work of type 1 and 2 could be easily accounted into cost associated with jump-outs (this will just add at most 2 to the multiplicative constant in Big-Oh notation of Lemma 5). So we only have to worry about work of type 3.

Definition 1 A block B_i is called *unchanged in Phase p* if $D_{p-1}(i) = D_p(i)$.

Let U_p be the set of indices of the unchanged blocks in Phase p , *i.e.*

$$U_p = \{i | D_{p-1}(i) = D_p(i)\}.$$

Note that, in the light of Lemma 1, U_p is uniquely determined by a fixed s and ρ . Let $u_p = |U_p|$. Then, in any run, work of type 3 costs at most $O(u_p n)$. We have following lemma.

Lemma 6 The expected cost of \mathcal{R} during Phase p is $O(m_p n \log n + u_p n)$.

9 Analyzing Adversary Cost

Now we are going to estimate the cost of \mathcal{A} in a phase. Consider two copies of a block B , B_1 and B_2 . Assume that \mathcal{A} has a evaders in B_1 , and \mathcal{R} has r evaders in B_2 . We also assume that $a \geq r$. Let ρ be a request sequence composed of positions in B . We allow any kind of movements for the \mathcal{A} -evaders in B_1 which satisfy ρ . The \mathcal{R} -evaders in B_2 will move according to our rules described earlier in subsection 5.2. It is important to note that neither the \mathcal{A} -evaders nor the \mathcal{R} -evaders will be allowed to jump out of their block. Then the following lemma holds.

Lemma 7 Let B_1 , B_2 , a , r and ρ be as above. Suppose that the \mathcal{R} -evaders in B_2 move T rounds combined when satisfying ρ . Then the \mathcal{A} -evaders in B_1 will move a distance of at least $\frac{aT}{3r} - \frac{2r}{3}$ combined.

Proof: Let's assume that \mathcal{A} is *lazy*.⁵ Further, to facilitate analysis, let's define following order for satisfying a request: First \mathcal{R} moves its evader in B_2 , if necessary, and next \mathcal{A} moves its evader in B_1 , if necessary.

While satisfying ρ , a \mathcal{R} -evader and an \mathcal{A} -evader may *meet* each other, *i.e.* both of them *momentarily* occupy the same position in their respective blocks while one or both of them are on the move. These meetings can be classified in three types using following definitions.

⁵ In response to a request ρ_i , a *lazy* adversary must only move the evader residing at ρ_i , if any, and no other evaders. This is not a real restriction. See [11, pp. 152].

Definition 2 A \mathcal{R} -evader is said to **catch** an \mathcal{A} -evader if one or both of them move during satisfaction of a request and then stop at the same position in their respective blocks.

Definition 3 A \mathcal{R} -evader is said to be **covering** an \mathcal{A} -evader during satisfaction of a request if they both remain motionless at the same position in their respective blocks.

Let x_1, x_2 denote any two \mathcal{R} -evaders and y_1, y_2 denote any two \mathcal{A} -evaders in following discussion.

1. **Jump-over:** x_2 cannot stop on a position in B_2 because there is x_1 on it, and x_1 is covering y_1 . We say x_2 *jumps over* y_1 .
2. **Leave:** x_1 and y_1 are residing at the same position in their respective blocks, but both leave the position when a request for it comes. We say x_1 *leaves* y_1 .
3. **Cross-over:** x_1 and y_1 cross each other when y_1 moves during satisfaction of a request and does not stop at x_1 's position. We say that y_1 *crosses-over* x_1 . Note that a moving x_1 can only catch or jump-over, it cannot cross-over owing to movement rules described in subsection 5.2. Moreover, when y_1 crosses-over x_1 , x_1 may or may not be covering another \mathcal{A} -evader.

Note that any meeting between a \mathcal{R} -evader and an \mathcal{A} -evader must be one of these three types. We consider *leave* as a meeting-type rather than its precursor *catch* event because leave implies some definite movement for the \mathcal{A} -evader.

\mathcal{A} -evaders move during leaves and cross-overs. Therefore we can prove lower bound on the cost of \mathcal{A} by estimating the same for the number of leaves and cross-overs. We do so using accounting method. Initialization of cash balance, associated money transactions, and relationships among \mathcal{R} - \mathcal{A} evaders (useful for book-keeping) are described below.

Initialization: Every \mathcal{R} -evader gets an initial amount of $2r$ dollars each, *i.e.* \mathcal{R} starts with $2r^2$ dollars overall. \mathcal{A} starts with infinite cash.

Transactions: During satisfaction of ρ , there are following monetary transactions:

- When x_1 catches y_1 , y_1 pays every \mathcal{R} -evader (including x_1) \$1 each.
- x_1 pays y_1 \$1 if x_1 is covering y_1 and x_2 jumps-over y_1 .
- x_1 pays y_1 \$1 and y_1 pays x_1 \$ $2r$ when the two leave each other.
- If y_1 crosses-over x_1 , x_1 pays y_1 \$1 while y_1 pays every \mathcal{R} -evader (including x_1) \$1 each.

Marriages: Every \mathcal{R} -evader has a unique *spouse* among \mathcal{A} -evaders at any given moment. We denote the spouse of x_1 by $s(x_1)$. Right at the start, $s(x_1) = y_1$ if x_1 and y_1 are at the same position in their respective blocks. Otherwise, we randomly select any *single* \mathcal{A} -evader as $s(x_1)$. Since $a \geq r$, we can ensure that

every \mathcal{R} -evader gets a unique spouse. Note that $a - r$ \mathcal{A} -evaders remain single. Unlike real life, these marital relationships undergo very dynamic changes: during satisfaction of ρ , whenever x_1 catches $s(x_2)$, x_1 and x_2 swap their spouses!

Whenever a \mathcal{R} -evader completes one round, it must meet each of the a \mathcal{A} -evaders at least once. Since \mathcal{R} -evaders complete T rounds combined, there must be then at least aT meetings between \mathcal{R} - \mathcal{A} evaders. Note that \mathcal{R} pays \$1 to \mathcal{A} per meeting. \mathcal{R} starts with $\$2r^2$ initial amount, it collects $\$3r$ for every catch and subsequent leave, and $\$r$ for every cross-over. Thus, the statement of the lemma follows readily if we prove that \mathcal{R} has a non-negative cash balance after every meeting. In fact, we prove stronger statement below.

Claim. Every \mathcal{R} -evader always has a non-negative cash balance.

Proof. Consider a \mathcal{R} -evader x_1 . It starts out with $\$2r$. As long as x_1 jumps-over or has cross-overs, it does not lose any money. Similarly, whenever x_1 catches an \mathcal{A} -evader y_1 and later leaves it, it collects $\$2r$ from y_1 . x_1 loses money only while it is covering y_1 . Therefore it suffices to show that x_1 is never in more than $\$2r$ loss while it is covering y_1 .

We focus our attention on the jump-overs that y_1 encounters while x_1 is covering y_1 . x_1 pays y_1 \$1 for each jump-over. Let's consider any other \mathcal{R} -evader, say x_2 , jumping over y_1 for m^{th} time since x_1 caught/covered y_1 , $m > 0$. This means x_2 has moved $\geq m - 2$ complete half-rounds since x_1 covered y_1 . Let's consider any such half-round of x_2 , and let $y_2 = s(x_2)$ when x_2 starts this half-round. Then one (or more) of the following must happen, and in each case, x_1 collects \$1.

1. x_2 catches y_2 .
2. x_2 and y_2 cross-over.
3. Another \mathcal{R} -evader catches y_2 .

Overall x_1 collects minimum $m - 2$ dollars owing to m jump-overs of x_2 . Summing for all $r - 1$ evaders, x_1 collects at least $M - 2r + 2$ dollars after M total jump-overs, and gives away only $\$M$, thus x_1 is in at most $2r - 2$ dollars loss. This proves the claim, and therefore, the lemma. \square

We use $T = 8b^2$ rounds as a threshold in marking procedure of \mathcal{R} . Also observe that the number of evaders r in any block is $< 2b$. We get following Corollary of Lemma 7.

Corollary 1 *Let B_1, B_2, a, r and ρ be as above. Suppose that the \mathcal{R} -evaders move $8b^2$ rounds combined when satisfying ρ . Then the \mathcal{A} -evaders in B_1 will move a distance of at least ab combined.*

Remark: If $r > a$, then it is possible that all the \mathcal{A} -evaders are covered by non-moving \mathcal{R} -evaders. Hence \mathcal{A} takes no steps while \mathcal{R} moves any number of rounds.

In the above we assumed that the \mathcal{A} -evaders stay in their block. But we do not have any control on them, they may change block too.

Definition 4 \mathcal{A} takes an \mathcal{A} -action in block B if either the \mathcal{A} -evaders move b steps inside B or one of them jumps out of B .

For example, in Corollary 1, we proved that \mathcal{A} takes at least a \mathcal{A} -actions in B if \mathcal{R} has $r \leq a$ evaders in B , the \mathcal{R} -evaders move $8b^2$ rounds, and all the \mathcal{A} -evaders and \mathcal{R} -evaders stay in B . According to our previous remark, if $a - r + 1$ \mathcal{A} -evaders jump out of B , then it is possible that no \mathcal{A} -evaders move inside B . But that would mean $a - r + 1$ \mathcal{A} -actions.

Lemma 8 Let block B has a \mathcal{A} -evaders and r \mathcal{R} -evaders at the start of a phase, and $a \geq r \geq 1$. \mathcal{R} -evaders are not allowed to jump in or jump out. If the request sequence forces the \mathcal{R} -evaders to move $8b^2$ rounds during the phase, then there are at least $a - r + 1$ \mathcal{A} -actions in block B during the phase.

Proof: Suppose that x \mathcal{A} -evaders jump out of block B during the phase. If $x \geq a - r + 1$, we are done. Otherwise, $x \leq a - r$. That means B always has at least $a - x \geq r$ \mathcal{A} -evaders. So using Corollary 1, we have at least $a - x$ more \mathcal{A} -actions. Total number of \mathcal{A} -actions, therefore, is $a - x + x = a \geq a - r + 1$. \square

Notice that so far we have been insisting that \mathcal{R} -evaders don't jump out of their block B . But we cannot assume that during the run of \mathcal{R} . What we know for sure though is that the auxiliary evaders in auxiliary blocks of B never jump around! Also rules of their movement are same as those of actual \mathcal{R} -evaders. This allows us to relate the mark of block B at the end of phase and \mathcal{A} -actions in B during the phase using Lemma 8.

Lemma 9 Let a be the number of \mathcal{A} -evaders in block B at the **Start of Phase** p . Let r be the number of \mathcal{R} -evaders in B at the **End of Phase** p . If $a > r$, then there are at least $a - r$ \mathcal{A} -actions in B during Phase p .

Proof: Due to our marking procedure, the mark of B , i.e. $m(B)$, at the end of Phase p is at most $r + 1$.⁶ Consider copy of B in adversary algorithm, and $AB_{m(B)}$, the $m(B)$ th auxiliary block of B in \mathcal{R} . $m(B)$ being the mark of B , the number of rounds in $AB_{m(B)}$ is at least $8b^2$. Since $a \geq r + 1 \geq m(B) \geq 1$, applying Lemma 8 on these two copies of B , we prove that the number of \mathcal{A} -actions in B during the phase is at least $a - m(B) + 1 \geq a - r$. \square

Now we are ready to compare the number of \mathcal{A} -actions with the expected cost of \mathcal{R} . Let's fix shift s . Let m_p denote the number of moving evaders and u_p denote the number of unchanged blocks in Phase p . Let a_p denote the number of \mathcal{A} -actions in Phase p by adversary. We will make following assumption for the proofs of Lemma 10 and Theorem 1.

Assumption: For every p and its corresponding U_p , following holds: If $i \in U_p$, then

$$|C_{p-1}(i) - C_p(i)| + |C_{p-1}(i) - D_p(i)| \geq 1. \quad (4)$$

⁶ In fact, it is one more than the number of evaders for all, but one, blocks!

Remark: Above assumption ensures nonzero \mathcal{A} -actions in an unchanged block. This is enough for competitiveness result since it counters the $O(n)$ work done by the \mathcal{R} -evaders in absence of any jump-in or jump-out of such block (see subsection 8.3). Otherwise consider a block B_i which is unchanged for consecutive phases $p_1, p_1 + 1, \dots, p_2$, and violates the assumption in each of them, *i.e.* $C_{p-1}(i) = C_p(i) = D_{p-1}(i) = D_p(i)$ for $p = p_1, p_1 + 1, \dots, p_2$.⁷ Let's further assume that \mathcal{A} -evaders don't move in or move out of B_i during any of these phases, since that ensures nonzero \mathcal{A} -actions. Then the contribution of B_i to \mathcal{A} -actions can only come from the movement of \mathcal{A} -evaders inside B_i . Let's suppose that \mathcal{R} -evaders move total of T rounds combined over all these $p_2 - p_1 + 1$ phases. Then, in the light of Lemma 7, competitive ratio of \mathcal{R} in B_i is $O(b/(1 - \frac{2r}{T})) = O(n^{\frac{2}{3}})$ if $T > 2r$, *i.e.* \mathcal{R} performs better than $O(n^{\frac{2}{3}} \log n)$ in B_i during these phases. In case there are fewer than $2r$ rounds, \mathcal{R} pays overhead of at most $O(rb + b^2) = O(n^{\frac{2}{3}})$ cost in B_i . It can be either absorbed in further jump-ins (or jump-outs) of block B_i in Phase $p_2 + 1$ or adjusted against the \mathcal{A} -actions of B_i in Phase $p_2 + 1$. If p_2 were the final phase, then it can be absorbed in the constant function I of equation (1). Thus, we are assuming worse scenario for analysis of \mathcal{R} with the assumption (4). Now we have following key lemma.

Lemma 10 *Fix the shift s . Let ρ be a request sequence for which \mathcal{R} has F phases. Then $\sum_{p=1}^F a_p + \frac{n}{7b} \geq \frac{1}{7} \sum_{p=1}^F (m_p + u_p)$.*

Proof: Recall, from the remark at the end of Lemma 1, we have equation (3)

$$m_p = \frac{1}{2} \sum_{i=1}^t |D_{p-1}(i) - D_p(i)|.$$

Similarly we can lower bound the number of \mathcal{A} -actions by the number of block changes

$$a_p \geq \frac{1}{2} \sum_{i=1}^t |C_{p-1}(i) - C_p(i)|. \quad (5)$$

But a_p is at least $\sum_{i=1}^t \max\{0, C_{p-1}(i) - D_p(i)\}$ by Lemma 9. Therefore,

$$a_p \geq \frac{1}{2} \sum_{i=1}^t |C_{p-1}(i) - D_p(i)|. \quad (6)$$

Simple triangle inequality, and inequalities (5), (6) and (4) imply

$$a_p \geq \frac{1}{4} \sum_{i=1}^t |C_p(i) - D_p(i)| \quad (7)$$

$$a_p \geq \frac{u_p}{4}. \quad (8)$$

⁷ Note that following must be true for Phase $p_1 - 1$ and Phase $p_2 + 1$: B_i is either *not* unchanged or it satisfies the assumption.

Once again, simple triangle inequality, and inequalities (7), (8), (6) and equation (3) imply

$$\begin{aligned}
6a_{p-1} + a_p &\geq \frac{1}{2} \sum_{i=1}^t |C_{p-1}(i) - D_{p-1}(i)| + u_{p-1} \\
&\quad + \frac{1}{2} \sum_{i=1}^t |C_{p-1}(i) - D_p(i)| \\
&\geq \frac{1}{2} \sum_{i=1}^t |D_{p-1}(i) - D_p(i)| + u_{p-1} \\
&= m_p + u_{p-1}.
\end{aligned} \tag{9}$$

Using equation (9), summing over all phases from 1 to F , and noting that $a_0 = u_0 = 0$ and $u_p \leq t = \frac{n}{b}$, we get

$$\begin{aligned}
\sum_{p=1}^F a_p + \frac{n}{7b} &\geq \frac{1}{7} \sum_{p=1}^F (6a_{p-1} + a_p) + \frac{n}{7b} \\
&\geq \frac{1}{7} \sum_{p=1}^F (m_p + u_p).
\end{aligned}$$

□

Next we determine the expected number of \mathcal{A} 's block changes and use that to estimate the expected number of \mathcal{A} -actions. The randomness is in the random choice of the shift s .

Lemma 11 *Let ρ be a request sequence, then the expected number of \mathcal{A} 's block changes is $\text{opt}(\rho) \cdot b^{-1}$.*

Proof: We assume that the \mathcal{A} -evaders always move one step at a time. This can be achieved by possibly dividing the motion of an evader into individual steps of length one, and if necessary, by switching role between evaders when one passes the other. Consider the indicator random variables $X_1, X_2, \dots, X_{\text{opt}(\rho)}$. $X_i = 1$ if in the i^{th} step the moving \mathcal{A} -evader changes block, otherwise it is zero. Then clearly $\Pr(X_i = 1) = b^{-1}$ for all i . So the expected number of block changes is $E[\sum_i X_i] = \sum_i \Pr(X_i = 1) = \text{opt}(\rho) \cdot b^{-1}$. □

Lemma 12 *For a request sequence ρ , the expected number of \mathcal{A} -actions is at most $2 \cdot \text{opt}(\rho) \cdot b^{-1}$*

Proof: By Lemma 11, the expected number of block changes during the satisfaction of ρ is $\text{opt}(\rho) \cdot b^{-1}$. The same number is clearly an upper bound for the number of other type of \mathcal{A} -actions, *viz.* work done inside a block. So the expected number of \mathcal{A} -actions is at most $2 \cdot \text{opt}(\rho) \cdot b^{-1}$. □

10 Proof of Theorem 1

We are ready to prove our main result, Theorem 1. Let ρ be a request sequence. Let us denote by m_p^s the number of moving \mathcal{R} -evaders in Phase p when the shift of the block system is s and the number of phases for shift s by t_s . Let u_p^s denote the number of unchanged blocks during Phase p for shift s . Random variable $R(\rho)_p^s$ will denote the cost of \mathcal{R} in Phase p with shift s and random variable $R(\rho)^s$ will denote the cost of \mathcal{R} on ρ with shift s .

By Lemma 12,

$$2 \cdot \text{opt}(\rho) \cdot b^{-1} \geq E[\text{Number of } \mathcal{A}\text{-actions}].$$

Lemma 10 implies that

$$E[\text{Number of } \mathcal{A}\text{-actions}] + \frac{n}{7b} \geq \frac{1}{7b} \sum_{s=1}^b \sum_{p=1}^{t_s} (m_p^s + u_p^s).$$

By Lemma 6, the latter expression is bounded below by

$$\frac{1}{7b} \sum_{s=1}^b \sum_{p=1}^{t_s} \left(E[R(\rho)_p^s] \frac{1}{O(n \log n)} \right).$$

By linearity of expectation, the above formula is

$$\frac{1}{O(bn \log n)} \sum_{s=1}^b E[R(\rho)^s].$$

Taking the average for shifts and using above inequalities, we have

$$2 \cdot \text{opt}(\rho) \cdot b^{-1} + \frac{n}{7b} \geq E[R(\rho)] \frac{1}{O(n \log n)}.$$

That is,

$$O\left(\frac{n \log n}{b}\right) \text{opt}(\rho) + O\left(\frac{n^2 \log n}{b}\right) \geq E[R(\rho)].$$

So the competitive ratio is $O(n^{\frac{2}{3}} \log n)$. □

Acknowledgment: The authors wish to thank Endre Szemerédi and Péter Hajnal for their helpful remarks and discussions.

References

1. Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 184–193, October 1996.

2. Y. Bartal. On approximating arbitrary metrics by tree metrics. In *Proceedings of 30th Annual ACM Symposium on Theory of Computing*, pages 161–168, May 1998.
3. Y. Bartal, A. Blum, C. Burch, and A. Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *Proceedings of 29th Annual ACM Symposium on Theory of Computing*, pages 711–719, May 1997.
4. Y. Bartal, B. Bollobás, and M. Mendel. A Ramsey-type theorem for metric spaces and its applications for metrical task systems and related problems. In *Proceedings of 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 396–405, October 2001.
5. Y. Bartal, M. Chrobak, and L. Larmore. A randomized algorithm for two servers on the line. *Information and Computation*, 158:53–69, 2000.
6. Y. Bartal and E. Koutsoupias. On the competitive ratio of the work function algorithm for the k -server problem. In *Proceedings of 17th Annual Symposium on Theoretical Aspects of Computer Science*, pages 605–613, February 2000.
7. W. Bein, M. Chrobak, and L. Larmore. The 3-server problem in the plane. In Jaroslav Nešetřil, editor, *Algorithms - ESA '99, 7th Annual European Symposium, Prague, Czech Republic, July 16-18, 1999, Proceedings*, volume 1643 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 1999.
8. S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11:2–14, 1994.
9. A. Blum, H. Karloff, Y. Rabani, and M. Saks. A decomposition theorem for task systems and bounds for randomized server problems. *SIAM Journal on Computing*, 30(5):1624–1661, 2000.
10. A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. *SIAM Journal on Computing*, 26:110–137, 1997.
11. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
12. A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task system. *Journal of the ACM*, 39:745–763, 1992.
13. A. Calderbank, E. Coffman, and L. Flatto. Sequencing problems in two server systems. *Mathematics of Operation Research*, 10:585–598, 1985.
14. M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4:172–181, 1991.
15. M. Chrobak and L. Larmore. An optimal algorithm for k servers on trees. *SIAM Journal on Computing*, 20:144–148, 1991.
16. B. Csaba. Note on the work function algorithm. *Acta Cybernetica*, 14:503–506, 2000.
17. B. Csaba and S. Lodha. A randomized online algorithm for the k -server problem on a line. Technical Report 2001-34, DIMACS, October 2001.
18. A. Fiat and G. Woeginger (Eds.). *Online Algorithms: State of the Art*. Springer-Verlag, 1998.
19. A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.
20. A. Fiat and M. Mendel. Better algorithms for unfair metrical task systems and applications. In *Proceedings of 32nd Annual ACM Symposium on Theory of Computing*, pages 725–734, May 2000.
21. S. Irani and A. Karlin. Online computation. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, pages 521–564. PWS Publishing Company, 1997.

22. A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11:542–571, 1994.
23. E. Koutsoupias and C. Papadimitriou. On the k -server conjecture. *Journal of the ACM*, 42(5):971–983, September 1995.
24. E. Koutsoupias and C. Papadimitriou. The 2-evader problem. *Information Processing Letters*, 57(5):249–252, 1996.
25. M. Manasse, L. McGeoch, and D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.
26. L. McGeoch and D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.
27. S. Seiden. A general decomposition theorem for the k -server problem. In Friedhelm Meyer auf der Heide, editor, *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 2001.
28. D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.