Noname manuscript No.
(will be inserted by the editor)

Tight Bounds for Double Coverage Against Weak Adversaries

Nikhil Bansal · Marek Eliáš · Łukasz Jeż · Grigorios Koumoutsos · and Kirk Pruhs

the date of receipt and acceptance should be inserted later

Abstract We study the Double Coverage (DC) algorithm for the k-server problem in tree metrics in the (h,k)-setting, i.e., when DC with k servers is compared against an offline optimum algorithm with $h \leq k$ servers. It is well-known that in such metric spaces DC is k-competitive (and thus optimal) for $h = k$. We prove that even if $k > h$ the competitive ratio of DC does not improve; in fact, it increases slightly as k grows, tending to $h+1$. Specifically, we give matching upper and lower bounds of $\frac{k(h+1)}{k+1}$ on the competitive ratio of DC on any tree metric.

Keywords k-server · weak adversaries · resource augmentation · double coverage

1 Introduction

We consider the k-server problem defined as follows. There are k servers in a given metric space. In each step, a request arrives at some point of the metric space and must be served by moving some server to that point. The goal is to minimize the total distance traveled by the servers.

The k-server problem was introduced by Manasse et al. [8] as a far reaching generalization of various online problems. The most well-studied of those is the paging

A preliminary version of this article appeared in the Proceedings of the *13th Workshop on Approximation and Online Algorithms* (WAOA 2015). This work was supported by NWO grant 639.022.211, ERC consolidator grant 617951, NCN grant DEC-2013/09/B/ST6/01538, NSF grants CCF-1115575, CNS-1253218, CCF-1421508, and an IBM Faculty Award.

N. Bansal, M. Eliáš, Ł. Jeż, G. Koumoutsos
Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. E-mail: {n.bansal, m.elias, l.jez, g.koumoutsos}@tue.nl

Ł. Jeż
Institute of Computer Science, University of Wrocław, ul. Joliot-Curie 15, 50-383 Wrocław, Poland. Tel.: +48 71 375 78 29 Fax: +48 71 375 78 01 E-mail: lje@cs.uni.wroc.pl

K. Pruhs
University of Pittsburgh, USA. E-mail: kirk@cs.pitt.edu

(caching) problem, which corresponds to k-server problem on a uniform metric space. Sleator and Tarjan [9] gave several k-competitive algorithms for paging and showed that this is the best possible ratio for any deterministic algorithm.

Interestingly, the k-server problem does not seem to get harder on more general metrics. The celebrated *k-server conjecture* states that a k-competitive deterministic algorithm exists for every metric space. In a breakthrough result, Koutsoupias and Papadimitriou [7] showed that the work function algorithm (WFA) is $(2k-1)$-competitive for every metric space, almost resolving the conjecture. The conjecture has been settled for several special metrics (an excellent reference is [2]). In particular for the line metric, Chrobak et al. [3] gave an elegant k-competitive algorithm called Double Coverage (DC). This algorithm was later extended and shown to be k-competitive for all tree metrics [4]. Additionally, in [1] it was shown that WFA is k-competitive for some special metrics, including the line.

The (h,k)-server problem: In this paper, we consider the (h,k)-setting, where the online algorithm has k servers, but its performance is compared to an offline optimal algorithm with $h \leq k$ servers. This is also known as the weak adversaries model [6], or the resource augmentation version of k-server. It is a salient point whether the algorithm knows the value of h. We assume that it does not, as the DC algorithm that we analyze does not utilize this value (and the same is true of WFA). Moreover, this assumption is more in the spirit of resource augmentation. Note that in general this distinction matters, as knowing h, an algorithm might decide to limit the number of servers it will use to serve the requests. The (h,k)-server setting turns out to be much more intriguing and is much less understood.

For the uniform metric (the (h,k)-paging problem), $k/(k-h+1)$-competitive algorithms are known [9] and no deterministic algorithm can achieve a better ratio. Note that this guarantee equals k for $h=k$, and tends to 1 as the ratio of the number of online to offline servers k/h becomes arbitrarily large. This shows that the weak adversaries model could give more accurate interpretation on the performance of online algorithms: The competitive ratio of k obtained in the classical setting grows with the number of servers, which could possibly mean that more servers worsen the performance of an algorithm. On the other hand, the ratio obtained in the (h,k) setting shows that the performance improves substantially when the number of servers grows. The same competitive ratio can also be achieved for the weighted caching problem [10] (and even the more general file caching problem [11], which is not a special case of the (h,k)-server problem).

However, unlike classical k-server, the underlying metric space seems to play an important role in the (h,k)-setting. Bar-Noy and Schieber (cf. [2, p. 175]) showed that, for the $(2,k)$-server problem on a line metric, no deterministic algorithm can be better than 2-competitive for any k. In particular, the ratio does not tend to 1 as k increases.

In fact, there is huge gap in our understanding of the problem, even for very special metrics. For example, for the line no guarantee better than h is known even when $k/h \to \infty$. On the other hand, the only lower bounds known are the result of Bar-Noy and Schieber mentioned above and a general lower bound of $k/(k-h+1)$ for any metric space with at least $k+1$ points (cf. [2] for both results). In particular,

no lower bound better than 2 is known for any metric space and any $h > 2$, if we let $k/h \to \infty$. The only general upper bound is due to Koutsoupias [6], who showed that WFA is $2h$-competitive[1] for the (h,k)-server problem on any metric. It is worth stressing that this is an upper bound for WFA that is oblivious of h and uses all of its k servers, and that ratio $2h-1$ can be attained by running WFA with h servers when this value is known to the algorithm.

The DC algorithm: This motivates us to consider the (h,k)-server problem on the line and more generally on trees. In particular, we consider the DC algorithm [3] originally defined for a line, and its generalization to trees [4]. We refer to both as DC, since the latter specializes to the former when the underlying tree is in fact a line. As understanding both the algorithm and its analysis for the line may be simpler and more insightful, we provide definitions of both variants. In both, we call an algorithm's server s adjacent to the request r if there are no algorithm's servers on the unique path between the locations of r and s, excluding the point where s is located. Note that there may be multiple servers in one location, satisfying this requirement — in such case, one of them is chosen arbitrarily as the adjacent server for this location, and the others are considered non-adjacent.

DC-Line: If the current request r lies outside the convex hull of current servers, serve it with the nearest server. Otherwise, we move the two servers adjacent to r towards it with equal speed until some server reaches r.

DC-Tree: Repeat the following until a server reaches the request r, constantly updating the set of adjacent servers: move all the servers adjacent to r towards r at equal speed. Note that the set of servers adjacent to r can change only when one of them reaches either a vertex of the tree or the request itself, which ends the move. We call the parts of the move between updates of the set of adjacent servers *elementary moves*.

There are several natural reasons to consider DC for line and trees. For paging (and weighted paging), all known k-competitive algorithms also attain the optimal ratio for the (h,k) version. This suggests that a k-competitive algorithm for the k-server problem might attain the "right" ratio in the (h,k)-setting. The only algorithm that satisfies this condition for a non-trivial metric is DC for trees, as well as WFA for the simpler case of a line. Of the two, DC has the advantage that it attains the optimum $k/(k-h+1)$-competitive ratio for the (h,k)-paging problem, when it is modelled as a star graph where requests appear in leaves, since it is equivalent to Flush-When-Full algorithm, as pointed out by Chrobak and Larmore [4]; see Appendix A for an explicit proof. As for WFA, all known upper bounds, including [6], bound the *extended cost* instead of the actual cost of the algorithm. Using this approach we can easily show that WFA is $(h+1)$-competitive for the line (cf. Appendix B).

Our Results: We show that the exact competitive ratio of DC on lines and trees in the (h,k)-setting is $\frac{k(h+1)}{(k+1)}$.

[1] Actually, [6] gives a stronger bound: $\mathrm{WFA}_k \leq 2h \cdot \mathrm{OPT}_h - \mathrm{OPT}_k + O(1)$, where the algorithm's subscripts specify how many servers they use.

Theorem 1 *The competitive ratio of DC is at least* $\frac{k(h+1)}{(k+1)}$*, even for a line.*

Note that for a fixed h, the competitive ratio worsens slightly as the number of online servers k increases. In particular, it equals h for $k = h$ and it approaches $h+1$ as $k \to \infty$.

Consider the seemingly trivial case of $h = 1$. If $k = 1$, clearly DC is 1-competitive. However, for $k = 2$ it becomes $4/3$ competitive, as we now sketch. Consider the instance where all servers are at $x = 0$ initially. A request arrives at $x = 2$, upon which both DC and offline move a server there and pay 2. Then a request arrives at $x = 1$. DC moves both servers there and pays 2 while offline pays 1. All servers are now at $x = 1$, and the instance repeats.

Generalizing this example to $(1,k)$ already becomes quite involved. Our lower bound in Theorem 1 for general h and k is based on an adversarial strategy obtained by a careful recursive construction.

We also give a matching upper bound.

Theorem 2 *For any tree, the competitive ratio of DC is at most* $\frac{k(h+1)}{(k+1)}$.

This generalizes the previous results for $h = k$ [3,4]. Our proof also follows similar ideas, but our potential function is more involved (it has three terms instead of two), and the analysis is more subtle. To keep the main ideas clear, we first prove Theorem 2 for the simpler case of a line in Section 3. The proof for trees is analogous but more involved, and is described in Section 4.

2 Lower Bound for the Line Metric

We now prove Theorem 1. We will describe an adversarial strategy S_k for the setting where DC has k servers and the offline optimum (adversary) has h servers, whose analysis establishes that the competitive ratio of DC is at least $k(h+1)/(k+1)$.

Roughly speaking (and ignoring some details), the strategy S_k works as follows. Let $I = [0, b_k]$ be the *working interval* associated with S_k. Let $L = [0, \varepsilon b_k]$ and $R = [(1-\varepsilon)b_k, b_k]$ denote the (tiny) *left front* and *right front* of I. Initially, all offline and online servers are located in L. The adversary moves all its h servers to R and starts requesting points in R, until DC eventually moves all its servers to R. The strategy inside R is defined recursively depending on the number of DC servers currently in R: if DC has i servers in R, the adversary executes the strategy S_i repeatedly inside R, until another DC server arrives there, at which point it switches to the strategy S_{i+1}. When all DC servers reach R, the adversary moves all its h servers back to L and repeats the symmetric version of the above instance until all servers move from R to L. This defines a *phase*. To show the desired lower bound, we recursively bound the online and offline costs during a phase of S_k in terms of costs incurred by strategies $S_1, S_2, \ldots, S_{k-1}$.

A crucial parameter of a strategy will be the *pull*. Recall that DC moves some server q_L closer to R if and only if q_L is the rightmost DC server outside R and a request is placed to the left of q_R, the leftmost DC server in R, as shown in Figure 1. In this situation q_R moves by δ to the left and q_L moves to the right by the same distance,

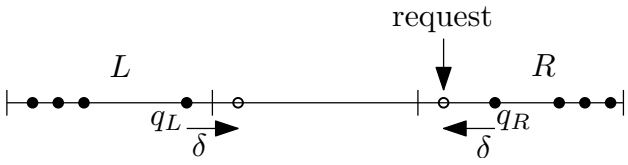


Fig. 1 DC server is pulled to the right by δ

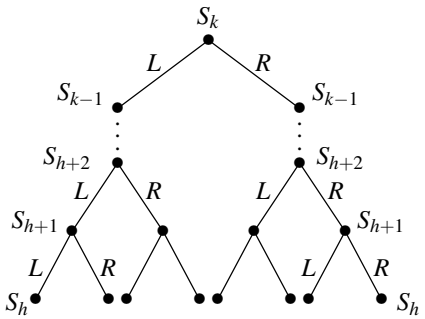


Fig. 2 Representation of strategies and the areas that they define using a binary tree.

and we say that the strategy in R exerts a *pull* of δ on q_L. We will be interested in the amount of pull exerted by a strategy during one phase.

Formal description: We now give a formal definition of the instance. We begin by introducing the quantities (that we bound later) associated with each strategy S_i during a single phase:

- d_i, lower bound for the cost of DC inside the working interval.
- A_i, upper bound for the cost of the adversary.
- p_i, P_i, lower resp. upper bound for the "pull" exerted on any external DC servers located to the left of the working interval of S_i. Note that, as will be clear later, by symmetry the same pull is exerted to the right.

For $i \geq h$, the ratio $r_i = \frac{d_i}{A_i}$ is a lower bound for the competitive ratio of DC with i servers against an adversary with h servers.

We now define the right and left front precisely. Let $\varepsilon > 0$ be a sufficiently small constant. For $i \geq h$, we define the size of working intervals for strategy S_i as $s_h := h$ and $s_{i+1} := s_i/\varepsilon$. Note that $s_k = h/\varepsilon^{k-h}$. The working interval for strategy S_k is $[0, s_k]$, and inside it we have two working intervals for strategies S_{k-1}: $[0, s_{k-1}]$ and $[s_k - s_{k-1}, s_k]$. We continue this construction recursively and the nesting of these intervals creates a tree-like structure as shown in Figure 2. For $i \geq h$, the working intervals for strategy S_i are called type-i intervals. Strategies S_i, for $i \leq h$, are special and are executed in type-h intervals.

Strategies S_i for $i \leq h$: For $i \leq h$, strategies S_i are performed in a type-h interval (recall this has length h). Let Q be $h+1$ points in such an interval, with distance 1 between consecutive points.

There are two variants of S_i that we call $\overrightarrow{S_i}$ and $\overleftarrow{S_i}$. We describe $\overrightarrow{S_i}$ in detail, and the construction of $\overleftarrow{S_i}$ will be exactly symmetric. At the beginning of $\overrightarrow{S_i}$, we ensure

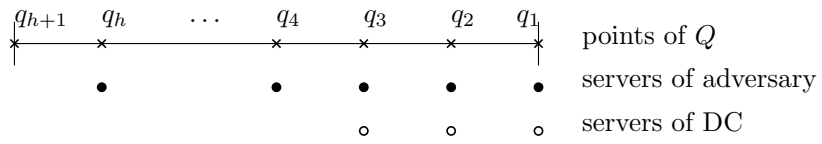


Fig. 3 The initial position for Strategy $\vec{S_3}$ (for $h \geq 3$), in which the adversary requests q_4, q_3, q_2, q_1. DC's servers move for a total of 6, exerting a pull of 1 in the process, only to return to the same position. The adversary's cost is 0 if $h > 3$ and 2 if $h = 3$: in such case, the adversary serves both q_4 and q_3 with the server initially located in q_3.

that DC servers occupy the rightmost i points of Q and adversary servers occupy the rightmost h points of Q as shown in Figure 3. The adversary requests the sequence $q_{i+1}, q_i, \ldots, q_1$. It is easily verified that DC incurs cost $d_i = 2i$, and its servers return to the initial position $q_i, \ldots, q_1$, so we can iterate $\vec{S_i}$ again. Moreover, a pull of $p_i = 1 = P_i$ is exerted in both directions.

As for the cost incurred by the adversary, we have $A_i = 0$, for $i < h$, as the offline servers do not have to move at all. For $i = h$, the offline can serve the sequence with cost 2, by using the server in q_h to serve request in q_{h+1} and then moving it back to q_h, therefore $A_h = 2$.

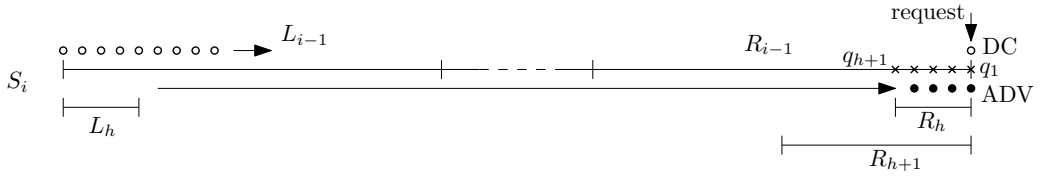
For strategy $\overleftarrow{S_i}$, we just number the points of Q in the opposite direction (q_1 will be leftmost and q_{h+1} rightmost). The request sequence, analysis, and assumptions about initial position are the same.

Strategies S_i for $i > h$: We define the strategy S_i for $i > h$, assuming that $S_1, \ldots, S_{i-1}$ are already defined. Let I denote the working interval for S_i. We assume that, initially, all offline and DC servers lie in the leftmost (or analogously rightmost) type-$(i-1)$ interval of I. Indeed, for S_k this is achieved by the initial configuration, and for $i < k$ we will ensure this condition before applying strategy S_i. In this case our phase consists of left-to-right step followed by right-to-left step (analogously, if all servers start in the rightmost interval, we apply first right-to-left step followed by left-to-right step to complete the phase).

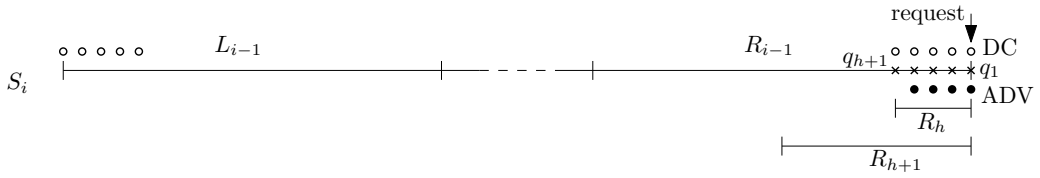
For each $h \leq j < i$, let L_j and R_j denote the leftmost and the rightmost type-j interval contained in I respectively.

Left-to-right step:

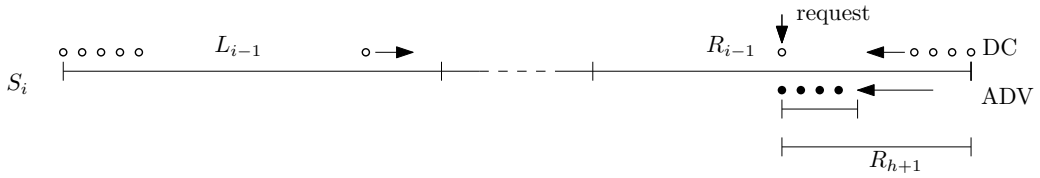
1. Adversary moves all its servers from L_{i-1} to R_h, specifically to the points $q_1, \ldots, q_h$ to prepare for the strategy $\vec{S_1}$. Next, point q_1 is requested, which forces DC to move one server to q_1, thus satisfying the initial conditions of $\vec{S_1}$. The figure below illustrates the servers' positions after these moves are performed.



2. For $j = 1$ to h: keep applying $\overrightarrow{S}_j$ to interval R_h until the $(j+1)$-th server arrives at the point q_{j+1} of R_h. (Recall that Figure 3 illustrates Strategy $\overrightarrow{S}_j$ for $j \leq h$.) Once it arrives there, complete the request sequence $\overrightarrow{S}_j$, so that DC servers will reside in points $q_{j+1}, \ldots, q_1$, ready for strategy $\overrightarrow{S}_{j+1}$. The figure below illustrates the servers' positions after all those moves (i.e., the whole outer loop, for $j = 1 \ldots, h$) are performed.



3. For $j = h+1$ to $i-1$: keep applying S_j to interval R_j until the $(j+1)$-th server arrives in R_j. To clarify, S_j stands for either $\overrightarrow{S}_j$ or $\overleftarrow{S}_j$, depending on the locations of servers within R_j. In particular, the first S_j for any j is $\overleftarrow{S}_j$. Note that there is exactly one DC server in the working interval of S_i moving toward R_j from the left: the other servers in that working interval are either still in L_{i-1} or not moving, since they are not adjacent to the request, or already in R_j. Since R_j is the rightmost interval of R_{j+1} and $L_{i-1} \cap R_{j+1} = \emptyset$, the resulting configuration is ready for strategy $\overleftarrow{S}_{j+1}$. The figure below illustrates the very beginning of this sequence of moves, for $j = h+1$, right after the execution of the first step (of this three-step description) of $\overleftarrow{S}_{j+1}$.



Right-to-left step: Same as Left-to-right, just replace $\overrightarrow{S}_j$ by $\overleftarrow{S}_j$, R_j intervals by L_j, and L_j by R_j.

Bounding Costs: We begin with a simple but useful observation that follows directly from the definition of *DC*. For any subset X of $i \leq k$ consecutive DC servers, let us call *center of mass* of X the average position of servers in X. We call a request *external* with respect to X, when it is outside the convex hull of X and *internal* otherwise.

Lemma 1 *For any sequence of internal requests with respect to X, the center of mass of X remains the same.*

Proof Follows trivially since for any internal request, DC moves precisely two servers towards it, by an equal amount in opposite directions. □

Let us derive bounds on d_i, A_i, p_i, and P_i in terms of these quantities for $j < i$. First, we claim that the cost A_i incurred by the adversary for strategy S_i during a

phase can be upper bounded as follows:

$$A_i \leq 2\left(s_i h + \sum_{j=1}^{i-1} A_j \frac{s_i}{p_j}\right) = 2s_i\left(h + \sum_{j=h}^{i-1} \frac{A_j}{p_j}\right) \tag{1}$$

In the inequality above, we take the cost for left-to-right step multiplied by 2, since left-to-right and right-to-left step are symmetric. The term $s_i h$ is the cost incurred by the adversary in the beginning of the step, when moving all its servers from the left side of I to the right. The costs $A_j \frac{s_i}{p_j}$ are incurred during the phases S_j for $j = 1,\ldots,i-1$, because A_j is an upper bound on the cost of the adversary during a phase of strategy S_j and $\frac{s_i}{p_j}$ is an upper bound on the number of iterations of S_j during S_i. This follows because S_j (during left to right phase) executes as long as the $(j+1)$-th server moves from left of I to right of I. It travels a distance of at most s_i and receives a pull of p_j during each iteration of S_j in R. Finally, the equality in (1) follows, as $A_j = 0$ for $j < h$.

Similarly, we bound the cost of DC from below. Let us denote $\delta := (1-2\varepsilon)$. The length of $I \setminus (L_{i-1} \cup R_{i-1})$ is δs_i and all DC servers moving from right to left have to travel at least this distance. Furthermore, as $\frac{\delta s_j}{P_j}$ is a lower bound for the number of iterations of strategy S_j, we obtain:

$$d_i \geq 2\left(\delta s_i i + \sum_{j=1}^{i-1} d_j \frac{\delta s_i}{P_j}\right) = 2\delta s_i\left(i + \sum_{j=1}^{i-1} \frac{d_j}{P_j}\right) \tag{2}$$

It remains to show the upper and lower bounds on the pull P_i and p_i exerted on external servers due to the (right-to-left step of) strategy S_i. Suppose S_i is being executed in interval I. Let x denote the closest DC server strictly to the left of I. Let X denote the set containing x and all DC servers located in I. During the right-to-left step of S_i, all requests are internal with respect to X. So by Lemma 1, the center of the mass of X remains unchanged. As i servers moved from right to left during right-to-left step of S_i, this implies that q should have been pulled to the left by the same total amount, which is at least $i\delta s_i$ and at most $i s_i$. Hence,

$$P_i := i s_i \qquad\qquad p_i := i\delta s_i \tag{3}$$

Due to a symmetric argument, during the left-to-right step, the same amount of pull is exerted to the right.

Now we are ready to prove Theorem 1.

Proof (of Theorem 1) The proof is by induction. In particular, we will show that the following holds for each $i \in [h,k]$:

$$\frac{d_i}{P_i} \geq 2i\delta^{i-h} \qquad \text{and} \qquad \frac{A_i}{p_i} \leq \frac{2(i+1)}{h+1}\delta^{-(i-h)} \tag{4}$$

Note that this claim already implies the theorem for $i = k$, since the competitive ratio r_k of DC_k satisfies the following inequality:

$$r_k \geq \frac{d_k}{A_k} \geq \frac{d_k/P_k}{A_k/p_k} \geq \frac{2k}{\frac{2(k+1)}{h+1}} \frac{\delta^{k-h}}{\delta^{-(k-h)}} = \frac{k(h+1)}{k+1}\delta^{2(k-h)} \quad .$$

Therefore, as $\delta = (1-2\varepsilon)$, it is easy to see that $r_k \to \frac{k(h+1)}{k+1}$ when $\varepsilon \to 0$:

Induction base ($i = h$): For the base case we have $a_h = 2$, $d_h = 2h$, and $p_h = P_h = 1$, so $\frac{d_h}{P_h} = 2h$ and $\frac{A_h}{p_h} = 2$, i.e., (4) holds.

Induction step ($i > h$): Using (2), (3), and induction hypothesis, we obtain

$$\frac{d_i}{P_i} \geq \frac{2\delta}{i}\left(i + \sum_{j=1}^{i-1}\frac{d_j}{P_j}\right) \geq \frac{2\delta}{i}\left(i + \sum_{j=1}^{i-1} 2j\delta^{j-h}\right) \geq \frac{2\delta}{i}\delta^{i-1-h}(i+i(i-1)) = 2i\delta^{i-h} \ ,$$

where the last inequality follows from the fact that $\sum_{j=1}^{i-1} 2j = i(i-1)$. Similarly, we prove the second part of (4). The first inequality follows from (1) and (3), the second from the induction hypothesis:

$$\begin{aligned}\frac{A_i}{p_i} &\leq \frac{2}{i\delta}\left(h + \sum_{j=h}^{i-1}\frac{A_j}{p_j}\right) \leq \frac{2}{i\delta}\left(h + \sum_{j=h}^{i-1}\frac{2(j+1)}{h+1}\delta^{-(j-h)}\right) \\ &\leq \frac{2}{i\delta}\delta^{-(i-1-h)}\left(\frac{h(h+1)+2\sum_{j=h}^{i-1}(j+1)}{h+1}\right) \\ &\leq \frac{2}{i\delta^{i-h}}\frac{i(i+1)}{h+1} = \frac{2(i+1)}{h+1}\delta^{-(i-h)} \ ,\end{aligned}$$

The last inequality follows from $2\sum_{j=h}^{i-1}(j+1) = i(i+1) - h(h+1)$. □

3 Upper Bound

In this section, we give an upper bound on the competitive ratio of DC that matches the lower bound from the previous section.

We begin by introducing some notation. We denote the optimal offline algorithm by OPT. For the current request r at time t, we let X and Y denote the configurations (i.e. the multisets of points in which their servers are located) of DC and OPT respectively before serving request r. Similarly, X' and Y' denote their corresponding configurations after serving r.

In order to prove our upper bound, we will define a potential function $\Phi(X,Y)$ such that

$$DC(t) + \Phi(X',Y') - \Phi(X,Y) \leq c \cdot OPT(t), \tag{5}$$

where $c = \frac{k(h+1)}{k+1}$ is the desired competitive ratio, and $DC(t)$ and $OPT(t)$ denote the cost incurred by DC and OPT at time t. Coming up with a potential that satisfies (5) is sufficient, as c-competitiveness follows from summing this inequality over time.

For a set of points A, let D_A denote the sum of all $\binom{|A|}{2}$ pairwise distances between points in A. Let $M \subseteq X$ be some fixed set of h servers of DC and $\mathscr{M}(M,Y)$ denote the minimum weight perfect matching between M and Y, where the weights are determined by distances in the metric space (i.e., tree). Abusing the notation slightly, we will denote by $\mathscr{M}(M,Y)$ both the matching and its cost. With that in mind, we let

$$\Psi_M(X,Y) := \frac{k(h+1)}{k+1} \cdot \mathscr{M}(M,Y) + \frac{k}{k+1} \cdot D_M \ .$$

Then the potential function is defined as follows:

$$\Phi(X,Y) = \min_M \Psi_M(X,Y) + \frac{1}{k+1}\cdot D_X$$
$$= \min_M \left(\frac{k(h+1)}{k+1}\cdot \mathscr{M}(M,Y) + \frac{k}{k+1}\cdot D_M \right) + \frac{1}{k+1}\cdot D_X \quad .$$

Note this generalizes the potential considered in [3,4] for the case of $h = k$. In that setting, all the online servers are matched and hence $D_M = D_X$ and is independent of M, and thus the potential above becomes k times that minimum cost matching between X and Y plus D_x. On the other hand in our setting, we need to select the right set M of DC servers to be matched to the offline servers based on minimizing $\Psi_M(X,Y)$.

Let us first give a useful property concerning minimizers of Ψ, which will be crucial later in our analysis. Note that $\Psi_M(X,Y)$ is not simply the best matching between X and Y, but also includes the term D_M which makes the argument slightly subtle.

Lemma 2 *Let X and Y be the configurations of DC and OPT and consider some fixed offline server at location $y \in Y$. There exists a minimizer M of Ψ that contains some DC server x which is adjacent to y. Moreover, there is a minimum cost matching $\mathscr{M}$ between M and Y that matches x to y.*

We remark that the adjacency in the lemma statement and the proof is defined as for the DC algorithm (cf. Section 1); specifically, as if there was a request at y's position. Moreover, we tote that the statement does not necessarily hold simultaneously for every offline server, but only for a single fixed offline server y.

Proof (of Lemma 2) Let M' be some minimizer of $\Psi_M(X,Y)$ and $\mathscr{M}'$ be some associated minimum cost matching between M' and Y. Let x' denote the online server currently matched to y in $\mathscr{M}'$ and suppose that x' is not adjacent to y. Let x denote the server in X adjacent to y on the path from y to x'.

We will show that we can always modify the matching (and M') without increasing the cost of Φ, so that y is matched to x. We consider two cases depending on whether x is matched or unmatched.

1. If $x \in M'$: Let y' denote the offline server which is matched to x in $\mathscr{M}'$. To create new matching $\mathscr{M}$, we swap the edges and match x to y and x' to y', see Figure 4. The cost of the edge connecting y in the matching reduces by exactly $d(x',y) - d(x,y) = d(x',x)$. On the other hand, the cost of the matching edge for y' increases by $d(x',y') - d(x,y') \leq d(x,x')$, due to triangle inequality. Thus, the new matching has no larger cost. Moreover, the set of matched servers does not change, i.e., $M = M'$, and hence $D_M = D_{M'}$, which implies that $\Psi_M(X,Y) \leq \Psi_{M'}(X,Y)$.
2. If $x \notin M'$: In this case, we set $M = M' \setminus \{x'\} \cup \{x\}$ and we form $\mathscr{M}$, where y is matched to x and all other offline servers are matched to the same server as in $\mathscr{M}'$. Now, the cost of the matching reduces by $d(x',y) - d(x,y) = d(x,x')$. Moreover, $D_M \leq D_{M'} + (h-1)\cdot d(x,x')$, as the distance of each server in $M' \setminus \{x'\}$ to x can

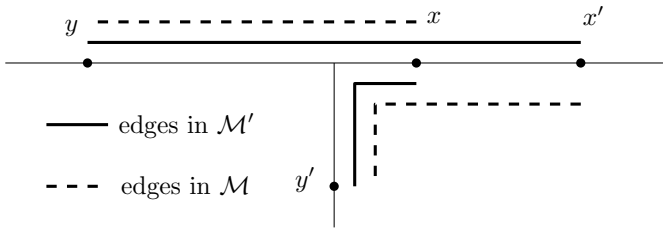


Fig. 4 Swapping of the matching edges in a tree.

be greater than the distance to x' by at most $d(x,x')$. This gives

$$\begin{aligned}\Psi_M(X,Y) - \Psi_{M'}(X,Y) &\leq -\frac{(h+1)k}{k+1} \cdot d(x,x') + \frac{k(h-1)}{k+1} \cdot d(x,x') \\ &= -\frac{2k}{k+1} \cdot d(x,x') < 0 \ ,\end{aligned}$$

and hence $\Psi_M(X,Y)$ is strictly smaller than $\Psi_{M'}(X,Y)$.

□

We are now ready to prove Theorem 2 for the line.

Proof Recall, that we are at time t and request r is arriving. We divide the analysis into two steps: (i) OPT serves r, and then (ii) DC serves r. As a consequence, whenever a server of DC serves r, we can assume that a server of OPT is already there.

In all the steps considered, M is the minimizer of $\Psi_M(X,Y)$ in the beginning of the step. It might happen that, after change of X,Y during the step, a better minimizer can be found. However, an upper bound for $\Delta\Psi_M(X,Y)$ is sufficient to bound the change in the first term of the potential function.

OPT moves: If OPT moves one of its servers by distance d to serve r, the value of $\Psi_M(X,Y)$ increases by at most $\frac{k(h+1)}{k+1}d$. As $OPT(t) = d$ and X does not change, it follows that

$$\Delta\Phi(X,Y) \leq \frac{k(h+1)}{k+1} \cdot OPT(t) \ ,$$

and hence (5) holds. We now consider the second step when DC moves.

DC moves: We consider two cases depending on whether DC moves a single server or two servers.

1. Suppose DC moves its rightmost server (the leftmost server case is identical) by distance d. Let y denote the offline server at r. By Lemma 2 we can assume that y is matched to the rightmost server of DC. Thus, the cost of the minimum cost matching between M and Y decreases by d. Moreover, D_M increases by exactly $(h-1)d$ (as the distance to rightmost server increases by d for all servers of DC). Thus, $\Psi_M(X,Y)$ changes by

$$-\frac{k(h+1)}{k+1} \cdot d + \frac{k(h-1)}{k+1} \cdot d = -\frac{2k}{k+1} \cdot d \ .$$

Similarly, D_X increases by exactly $(k-1)d$. This gives us that

$$\Delta\Phi(X,Y) \leq -\frac{2k}{k+1}\cdot d + \frac{k-1}{k+1}\cdot d = -d \ .$$

As $DC(t) = d$, this implies that (5) holds.

2. We now consider the case when DC moves 2 servers x and x', each by distance d. Let y denote the offline server at the request r. By Lemma 2 applied to y, we can assume that M contains at least one of x or x', and that y is matched to one of them (say x) in some minimum cost matching $\mathscr{M}$ of M to Y.
We note that D_X decreases by precisely $2d$. In particular, the distance between x and x' decreases by $2d$, and for any other server of $X \setminus \{x,x'\}$ its total distance to other servers does not change. Moreover, $DC(t) = 2d$. Hence, to prove (5), it suffices to show

$$\Delta\Psi_M(X,Y) \leq -\frac{k}{k+1}\cdot 2d \ . \tag{6}$$

To this end, we consider two sub-cases.

(a) *Both x and x' are matched:* In this case, the cost of the matching $\mathscr{M}$ does not increase as the cost of the matching edge (x,y) decreases by d and the move of x' can increase the cost of the matching by at most d. Moreover, D_M decreases by precisely $2d$ (due to x and x' moving closer). Thus, $\Delta\Psi_M(X,Y) \leq -\frac{k}{k+1}\cdot 2d$, and hence (6) holds.

(b) *Only x is matched (to y) and x' is unmatched:* In this case, the cost of the matching $\mathscr{M}$ decreases by d. Moreover, D_M can increase by at most $(h-1)d$, as x can move away from each server in $M \setminus \{x\}$ by distance at most d. So

$$\Delta\Psi_M(X,Y) \leq -\frac{(h+1)k}{k+1}\cdot d + \frac{k(h-1)}{k+1}\cdot d = -\frac{2k}{k+1}\cdot d \ ,$$

i.e., (6) holds.

□

4 Extension to Trees

The proof for trees is similar to the one in the previous section. The main difference is that the set of servers adjacent to the request can now be arbitrary (i.e., it no longer contains at most two servers) and that it can change as the move is executed, see Figure 5. To cope with this, we analyze elementary moves, as did Chrobak and Larmore [4]. Recall that an elementary move is a part of the move between successive updates of the set of servers adjacent to the request; consequently, this set remains fixed during such move.

Proof (of Theorem 2) We use the same potential as before, i.e, we let

$$\Psi_M(X,Y) := \frac{k(h+1)}{k+1}\cdot \mathscr{M}(M,Y) + \frac{k}{k+1}\cdot D_M \ ,$$

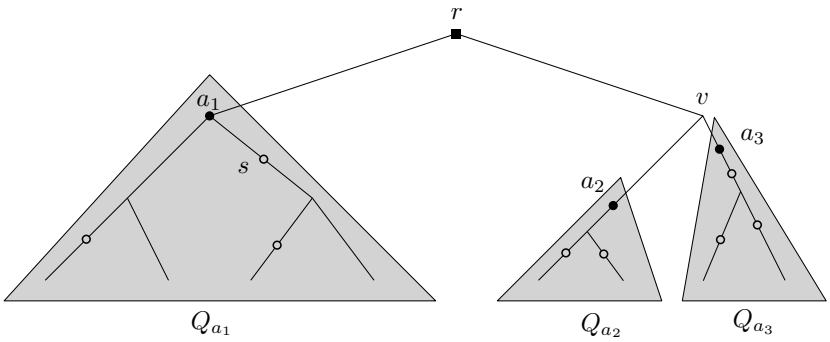


Fig. 5 Beginning of elementary move: server a_1 just covered server s removing him from the set of servers adjacent to request r. Servers a_1, a_2, and a_3 will move towards r, until a_3 reaches subroot v removing a_2 from the list of adjacent servers and completing thereby this elementary move.

and define

$$\Phi(X,Y) = \min_M \Psi_M(X,Y) + \frac{1}{k+1} \cdot D_X \ .$$

To prove the theorem, we show that for any time t the following holds:

$$DC(t) + \Phi(X',Y') - \Phi(X,Y) \leq c \cdot OPT(t), \tag{7}$$

where $c = \frac{k(h+1)}{k+1}$.

As in the analysis for the line, we split the analysis in two parts: (i) OPT serves r, and then (ii) DC serves r. As a consequence, whenever a server of DC serves r, we can assume that a server of OPT is already there.

OPT moves: If OPT moves a server by distance d, only the matching cost is affected in the potential function, and it can increase by at most $d \cdot k(h+1)/(k+1)$. Therefore

$$\Delta\Phi(X,Y) \leq \frac{k(h+1)}{k+1} \cdot OPT(t) \ ,$$

and hence (7) holds.

DC moves: Instead of focusing on the whole move done by DC to serve request r, we prove that (7) holds for each elementary move.

Consider an elementary move where q servers are moving by distance d. Let A denote the set of servers adjacent to r. Imagine that the tree is rooted at r, and let, for all $a \in A$, Q_a denote the subset of X (i.e., DC servers) that are located in the subtree rooted at a's location, including that point a, see Figure 5. We set $q_a := |Q_a|$ and $h_a := |Q_a \cap M|$. Finally, let $A_M := A \cap M$. By Lemma 2, we can assume that one of the servers in A is matched to the OPT's server in r, which implies

$$\Delta\mathscr{M}(M,Y) \leq (|A_M| - 2) \cdot d \ .$$

In order to calculate the change in D_X and D_M, it is convenient to consider the moves of active servers sequentially rather than simultaneously.

We start with D_X. Clearly, each $a \in A$ moves further away from $q_a - 1$ servers in X by distance d and gets closer to the remaining $k - q_a$ ones by the same distance. Thus, the change of D_X associated with a is $(q_a - 1 - (k - q_a))d = (2q_a - k - 1)d$. Therefore we have

$$\Delta D_X = \sum_{a \in A} (2q_a - k - 1)d = (2k - q(k+1))\, d \ ,$$

as $\sum_{a \in A} q_a = k$.

Similarly, for D_M, we first note that it can change only due to moves of servers in A_M. Specifically, each $a \in A_M$ moves further away from $h_a - 1$ servers in M and gets closer to the remaining $h - h_a$ of them. Thus, the change of D_M associated with a is $(2h_a - h - 1)d$. Therefore we have

$$\Delta D_M = \sum_{a \in A_M} (2h_a - h - 1)d \leq (2h - |A_M|(h+1))\, d \ ,$$

since $\sum_{a \in A_M} h_a \leq \sum_{a \in A} h_a = h$.

Using above inequalities, we see that the change of potential is at most

$$\begin{aligned}
\Delta \Phi(X,Y) &\leq \frac{k(h+1)d}{k+1}(|A_M| - 2) + \frac{k \cdot d}{k+1}(2h - |A_M|(h+1)) + \frac{d}{k+1}(2k - q(k+1)) \\
&\leq \frac{d}{k+1}\big(k(h+1)(|A_M| - 2) + k\,(2h - |A_M|(h+1)) + 2k - q(k+1)\big) \\
&= \frac{d}{k+1}(-q(k+1)) = -q \cdot d \ ,
\end{aligned}$$

since

$$\begin{aligned}
&k(h+1)(|A_M| - 2) + k(2h - |A_M|(h+1)) + 2k \\
&= -2k(h+1) + k(h+1)|A_M| - |A_M|k(h+1) + 2kh + 2k \\
&= -2k(h+1) + 2k(h+1) = 0
\end{aligned}$$

Thus, (7) holds, as $DC(t) = q \cdot d$. □

References

1. Bartal, Y., Koutsoupias, E.: On the competitive ratio of the work function algorithm for the k-server problem. Theor. Comput. Sci. **324**(2–3), 337–345 (2004)
2. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press (1998)
3. Chrobak, M., Karloff, H.J., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM J. Discrete Math. **4**(2), 172–181 (1991)
4. Chrobak, M., Larmore, L.L.: An optimal on-line algorithm for k-servers on trees. SIAM J. Comput. **20**(1), 144–148 (1991). DOI 10.1137/0220008
5. Chrobak, M., Larmore, L.L.: The server problem and on-line games. In: On-line Algorithms, volume 7 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pp. 11–64. AMS/ACM (1992)
6. Koutsoupias, E.: Weak adversaries for the k-server problem. In: Proc. of the 40th Symp. on Foundations of Computer Science (FOCS), pp. 444–449 (1999)
7. Koutsoupias, E., Papadimitriou, C.H.: On the k-server conjecture. J. ACM **42**(5), 971–983 (1995)

8. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for server problems. J. ACM **11**(2), 208–230 (1990)
9. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985). DOI 10.1145/2786.2793
10. Young, N.E.: The k-server dual and loose competitiveness for paging. Algorithmica **11**(6), 525–541 (1994). DOI 10.1007/BF01189992
11. Young, N.E.: On-line file caching. Algorithmica **33**(3), 371–383 (2002). DOI 10.1007/s00453-001-0124-5. Journal version of [1998].

A Analysis of DC for Paging

The paging problem is the special case of k-server on a uniform metric. It is also equivalent (up to a constant additive term) to the k-server problem on a star graph, where all edges have weight $\frac{1}{2}$ and requests appear at the leaves. This connection was pointed out by Chrobak and Larmore [4], who also noticed that DC-Tree can be interpreted as Flush-When-Full (FWF). It thus follows that it is $\frac{k}{k-h+1}$-competitive in the (h,k)-setting. As we are not aware an explicit proof of this fact, we give one that uses a potential function.

Let X and Y denote the configurations of DC and OPT respectively. Note that any server of DC can only be at the root or at a leaf, and servers of OPT can only be at leaves. Let ℓ denote the number of DC servers at the root.

We define the potential function as follows:

$$\Phi(t) = \frac{-k-h+1}{2(k-h+1)}\ell + \frac{k}{k-h+1}|Y \setminus X|$$

Analysis: We consider the moves of OPT and DC separately. We assume that, whenever a point is requested, first OPT moves a server there and then DC moves its servers.

Offline moves: When OPT moves any single server from one leaf to another, it pays 1. Clearly, ℓ does not change, and $|Y \setminus X|$ can increase by at most one. Thus, $\Delta\Phi \leq \frac{k}{k-h+1} = \frac{k}{k-h+1} \cdot OPT$.

DC moves: Let us now consider moves of DC. We distinguish between two cases depending on the value of ℓ:

- $\ell > 0$: In this case, DC moves one server from the root to the requested leaf, paying $1/2$. Clearly, both ℓ and $|Y \setminus X|$ decrease by 1. Thus,

$$\Delta\Phi = \frac{k+h-1}{2(k-h+1)} - \frac{k}{k-h+1} = \frac{-k+h-1}{2(k-h+1)} = -\frac{1}{2} \ ,$$

  and hence $DC + \Delta\Phi = 0$.
- $\ell = 0$: In this case, DC moves all the servers from the leaves toward the root (and then we go to the case above). In that case DC occurs a cost of $k/2$. Let us call a the number of online servers that coincide with servers of OPT before the move of DC. Then ℓ is increasing by k while $|Y \setminus X|$ increases by a. We get that

$$\Delta\Phi = \frac{-k-h+1}{2(k-h+1)}k + \frac{k}{k-h+1}a \ .$$

  Observe that $a \leq h-1$, as OPT already has a server covering the current request (and DC does not). Thus we can upper bound $\Delta\Phi$ as follows:

$$\Delta\Phi \leq \frac{-k-h+1+2(h-1)}{2(k-h+1)}k = \frac{-k+h-1}{2(k-h+1)}k = -\frac{k-h+1}{2(k-h+1)}k = -\frac{k}{2} \ .$$

  Overall we get that $DC + \Delta\Phi \leq \frac{k}{2} - \frac{k}{2} = 0$.

B Proof of $(h+1)$-competitiveness of WFA

We consider the WFA on the line in the (h,k) setting. Specifically we show that WFA with k servers is $(h+1)$-competitive against an adversary with h servers, as an immediate consequence of results shown in [1],[6].

Most known upper bounds for the WFA do not bound the algorithm's actual cost directly. Instead, they bound its *extended cost*, defined as the maximum increase the value of the work function for any single configuration. To define it formally, we use the following notation: $w_t(X)$ is the value of the work function of configuration X at time t, WFA_i and OPT_i the overall cost incurred by WFA and OPT of i servers respectively. Then, if M denotes the metric space, the extended cost at time t is defined as follows:

$$\text{ExtCost}(t) = \max_{X \in M^k} \{w_t(X) - w_{t-1}(X)\} \ ,$$

and the total extended cost of a sequence of requests is

$$\text{ExtCost} = \sum_t \text{ExtCost(t)} \ .$$

As with WFA and OPT, we let ExtCost_i denote the total extended cost over configurations of i servers. The extended cost satisfies the following inequality, as shown by Chrobak and Larmore [5]:

$$\text{WFA}_i + OPT_i \leq \text{ExtCost}_i \ . \tag{8}$$

In [1] it was shown that WFA with h servers is h-competitive in the line by proving the following inequality:

$$\text{ExtCost}_h \leq (h+1)OPT_h + O(1) \tag{9}$$

Moreover, it is known that the extended cost is a non-increasing function of the number of servers [6], which implies

$$\text{ExtCost}_k \leq \text{ExtCost}_h \tag{10}$$

for all request sequences.

Putting (8), (9) and (10) together, we get

$$\text{WFA}_k + OPT_k \leq \text{ExtCost}_k \leq \text{ExtCost}_h \leq (h+1)OPT_h + O(1) \ ,$$

which implies that WFA_k is $(h+1)$-competitive. In fact, a slightly stronger inequality holds:

$$\text{WFA}_k \leq (h+1)OPT_h - OPT_k + O(1) \ .$$