Tight bounds for Double Coverage against weak adversaries*

Nikhil Bansal[1], Marek Eliáš[1], Łukasz Jeż[1,2], Grigorios Koumoutsos[1], and Kirk Pruhs[3]

[1] Eindhoven University of Technology, Netherlands
{n.bansal,m.elias,l.jez,g.koumoutsos}@tue.nl
[2] Institute of Computer Science, University of Wrocław, Poland
[3] University of Pittsburgh, USA
kirk@cs.pitt.edu

Abstract. We study the Double Coverage (DC) algorithm for the k-server problem 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 DC is k-competitive for $h = k$. We prove that even if $k > h$ the competitive ratio of DC does not improve; in fact, it increases up to $h+1$ as k grows. In particular, we show matching upper and lower bounds of $\frac{k(h+1)}{k+1}$ on the competitive ratio of DC on any tree metric.

1 Introduction

We consider the k-server problem defined as follows. There is a metric space and k servers located on points in it. In each step, a request arrives at one of the points in the 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 defined by Manasse et al. [7] as a far reaching generalization of various online problems. The most well-studied of these is the paging (caching) problem, which corresponds to k-server on a uniform metric space. Sleator and Tarjan [8] 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, and the celebtrated *k-server conjecture* states that a k-competitive algorithm exists for every metric space. In a breakthrough result, Koutsoupias and Papadimitriou [6] 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.

* Supported by NWO grant 639.022.211 and ERC consolidator grant 617951.

(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 [5], or the resource augmentation version of k-server. 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 [8] 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. The same competitive ratio can also be achieved for the weighted caching problem [9].

However, unlike for k-server, the underlying metric space seems to play a very important role in the (h,k)-setting. Bar-Noy and Schieber (see [2], page 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 a 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 [5], who showed that WFA is at most $2h$-competitive[1] for the (h,k)-server problem on any metric[2].

The DC algorithm: This situation 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], defined as follows.

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: We move all the servers adjacent to r towards it at equal speed until some server reaches r. If there are multiple adjacent servers at the same location, we move one of them arbitrarily. (Note that the set of adjacent servers can change during the move, and is constantly updated.)

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 k-competitive algorithms for the k-server on the line might attain the "right" ratio for the (h,k)-setting. DC is the only (other than WFA) deterministic k-server algorithm known for

[1] Actually [5] shows a slightly stronger upper bound $\mathrm{WFA}_k \leq 2h\mathrm{OPT}_h - \mathrm{OPT}_k + \text{const}$ where OPT_k and OPT_h are the optimal cost using k and h servers respectively.

[2] If the online algorithm knows h, it can simply disable its $k-h$ extra servers and be $2h-1$ competitive (which is slightly better than $2h$). However, Koutsoupias (and also us) consider the setting where the online algorithm does not know h.

the line and trees. Moreover, DC obtains the optimum $k/(k-h+1)$-competitive ratio for the (h,k)-paging problem[3].

It seems plausible that WFA might perform very well for lines and trees as k increases, but no $o(h)$ bound is known. Most known upper bounds, including [5], 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[4].

Our Results: We determine the exact competitive ratio of DC on lines and trees in the (h,k)-setting.

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 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[5]. 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.

Next, we 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

We now prove Theorem 1. We will describe an adversarial strategy S_k for the setting where DC has k servers and the offline has h servers, and show that competitive ratio of DC can be made arbitrarily close to $k(h+1)/(k+1)$.

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 where the requests will arrive. Let $L=[0,\epsilon b_k]$ and $R=[(1-\epsilon)b_k, b_k]$ denote the (tiny) *left front* and *right front* in I. Initially, all the offline and online servers are located in L.

[3] This does not seem to be known, so we give a proof in the Appendix.

[4] In [1] it is shown that for the line $\text{ExtCost}_h \le (h+1)\ \text{OPT}_h +$ const. Moreover in [5] the monotonicity of extended cost was proven: $\text{ExtCost}_k \le \text{ExtCost}_h$. Using same arguments as in [5] it follows that $\text{WFA}_k \le (h+1)\text{OPT}_h - \text{OPT}_k +$ const.

[5] 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.

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. Roughly, if there are i DC servers in R, the adversary executes the strategy S_i repeatedly in the region R, until another DC server moves there, at which point it switches to the strategy S_{i+1}. When all the 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 will recursively bound the online and offline costs incurred during a phase of S_k in terms of costs incurred by strategies $S_1, S_2, \ldots, S_{k-1}$.

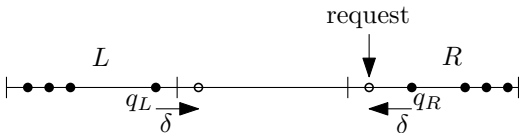

Fig. 1. DC server is pulled to the right by δ

A crucial parameter of a strategy will be the *pull*. Recall that DC moves some server q_L closer to R iff q_L is the rightmost DC server outside R and a request is placed to the left of the leftmost DC server q_R 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, and we say that the instance in R exerts a *pull* of δ on 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 defining the following quantities associated with each strategy S_i during a single phase:

- d_i, lower bound for the distance moved by DC.
- A_i, upper bound for the distance moved by ADV.
- 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 $r_i = \frac{d_i}{A_i}$ would be a lower bound for the competitive ratio of DC with i servers against 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.

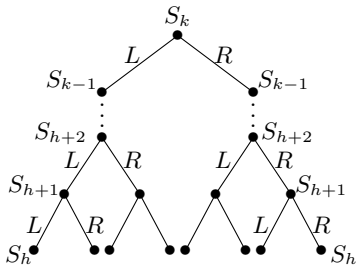

Fig. 2. Respresentation of strategies and the areas that they define using a binary tree.

It remains to specify the strategies S_i.

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

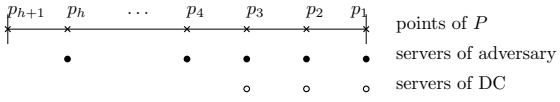

Fig. 3. Strategy $\vec{S}_3$, where $h \geq 3$.

There are two variants of S_i that we call $\vec{S}_i$ and $\overleftarrow{S}_i$. We describe $\vec{S}_i$ in detail, and the construction of $\overleftarrow{S}_i$ will be exactly symmetric. At the beginning of $\vec{S}_i$, we will ensure that DC servers occupy rightmost i points of P and offline servers occupy the rightmost h points of P as shown in Figure 3. The adversary requests the sequence $p_{i+1}, p_i, \ldots, p_1$. It is easily verified that DC incurs cost $d_i = 2i$, and its servers will return to the initial position $p_i, \ldots, p_1$, so we can iterate $\vec{S}_i$ again. Moreover a pull of $p_i = 1 = P_i$ is exerted in both directions.

For $i < h$, the adversary does not have to move at all and $A_i = 0$. For $i = h$, the offline can serve the sequence with cost $A_h = 2$, by using the server p_h to serve p_{h+1} and then moving it back to server p_h.

For strategy $\overleftarrow{S}_i$ we just number the points of P in the opposite direction (p_1 will be leftmost and p_{h+1} rightmost). Then 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 defined. Let I denote the working interval for S_i. We assume that, initially, all the offline servers and the 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 the S_i strategy. In this case our phase consists of left-to-right step followed by right-to-left step (analogously, if all the servers start in the rightmost interval, we apply first right-to-left step followed by left-to-right step to complete the phase).

Let L_j and R_j denote the leftmost and the rightmost type-j interval contained in I, for $h \leq j < i$.

Left-to-right step:

1. Adversary moves all its servers from L_{i-1} to R_h, specifically to the points $p_1, \ldots, p_h$ to prepare for the strategy $\overrightarrow{S_1}$. Next, point p_1 is requested which forces DC to move one server to p_1 and initial conditions of $\overrightarrow{S_1}$ are satisfied.
2. For $j = 1$ to h: apply $\overrightarrow{S_j}$ to interval R_h until $(j+1)$-th server arrives to point p_{j+1} in R_h. After server $j+1$ arrives, we finish the already started request sequence of S_j, so that DC servers will be lined in points $p_{j+1}, \ldots, p_1$ — ready for strategy S_{j+1}.
3. For $h < j < i$: apply S_j to interval R_j until $(j+1)$-th server arrives to R_j. Note, that this was the only DC server moving from L_{i-1} towards R_j. The rest are either still in L_{i-1} or in R_j. Since R_j is the rightmost interval of R_{j+1} and $L_{i-1} \cap R_{j+1} = \emptyset$, our configuration is ready for strategy 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 by an equal amount in opposite directions. □

Let us derive values d_i, A_i, p_i, and P_i assuming that they were already computed for all $j < i$. We claim that the offline cost A_i for strategy S_i during a phase can be upper bounded as follows.

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

The term $2s_i h$ follows as offline initially moves the h serves from left of I to right of I and the then back. 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 offline cost 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 distance at most s_i and feels a pull of p_j while S_j is executed in R. The equality above follows, as $A_j = 0$ for $j < h$.

We now lower bound the DC cost. 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 executing in interval I. Let q denote the closest DC server strictly to the left of I. Let Q denote the set containing q and all DC servers located in I. The crucial point is, that during the right-to-left step of S_i all requests look internal with respect to Q. So by Lemma 1, the center of the mass of these servers stays unchanged. As i servers moved from right to left by distance s_i 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 is_i.

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

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

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

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

Setting $i = k$, this implies the theorem as the competitive ratio r_k satisfies

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

As $\delta = (1 - 2\varepsilon)$, choosing $\varepsilon \ll 1/(k-h)$ small enough δ can be made arbitrarily close to 1, which implies the result.

Induction base $i = h$. For the base case we have the exact values of a_h and d_h, and, in particular, $\frac{d_h}{P_h} = 2h$ and $\frac{A_h}{p_h} = 2$.

Induction step $i > h$. Using (1), (2), and (3) we obtain:

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

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

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

3 Upper Bound

In this section, we show tightness of the lower bound from the previous section. By OPT we denote the optimal offline algorithm.

Let r be a request issued at time t. Let X denote configuration of DC (i.e. the set of points in the line where DC servers are located) and Y configuration of adversary before serving request r. Similarly, let X' and Y' be the corresponding configurations after serving r. In order to prove our upper bound, we 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.

Let $M \subseteq X$ be some fixed set of h servers of DC and $\mathcal{M}(M, Y)$ denote the cost of the minimum weight perfect matching between M and Y. We denote

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

Here, for a set of points A, D_A denotes the sum of all $\binom{|A|}{2}$ pairwise distances between points in A. The potential function is defined as follows:

$$\begin{aligned}\Phi(X, Y) &= \min_M \Psi_M(X, Y) + \frac{1}{k+1} \cdot D_X \\ &= \min_M \left(\frac{k(h+1)}{k+1} \cdot \mathcal{M}(M, Y) + \frac{k}{k+1} \cdot D_M\right) + \frac{1}{k+1} \cdot D_X \ .\end{aligned}$$

As opposed to analysis of the $h = k$ setting where all DC servers are matched, we need to select the right set M of DC servers to minimize whole $\Psi_M(X, Y)$.

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. We prove this lemma directly for trees, since it will be also useful in the following section.

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 $\mathcal{M}$ between M and Y that matches x to y*[6]*.*

Proof. Let M' be some minimizer of $\Psi_M(X,Y)$, and $\mathcal{M}'$ be some associated minimum cost matching between M' and Y. Let x' denote the online server currently matched to y in $\mathcal{M}'$. We denote x the adjacent server to y, in 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.

1. If $x \in M'$: Let us call y' the offline server which is matched to x in $\mathcal{M}'$. We swap the edges and match x to y and x' to y'. 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 $d(x',y') - d(x,y') \leq d(x,x')$. Thus, the new matching has no larger cost. Moreover, the set of matched servers $M = M'$ 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 $\mathcal{M}$, where y is matched to x and all other offline servers are matched to the same server as in $\mathcal{M}'$. Now, the cost of the matching reduces by $d(x',y) - d(x,y) = d(x,x')$ and $D_M \leq D_{M'} + (h-1) \cdot d(x,x')$ (as the distance of each server in $M' \setminus \{x'\}$ to x can 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) the offline serves r and then (ii) the online serves it. As a consequence, whenever a server of DC serves r, we can assume that a server of OPT is already there.

[6] We remark that this property does not hold (simultaneously) for every offline server, but only for a single fixed offline server y.

For all following steps considered, M will be 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, better minimizer can be found. However, upper bound for $\Delta\Psi_M(X,Y)$ will still be sufficient to bound the change in the first term of the potential function.

Offline moves: If offline 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 the minimizer 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 $\mathcal{M}$ 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 $\mathcal{M}$ does not go up as the cost of the matching edge (x,y) decreases by d and the cost of matching edge to x' can increase 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 $\mathcal{M}$ 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

We now consider tree metrics. Specifically, we prove Theorem 2. Part of the analysis carries over from the previous section. We use the same potential function as for the line. Observe that Lemma 2 holds for trees as well: We only used the triangle inequality and the fact that there exists a unique path between any two points.

Proof (of Theorem 2). The analysis of the step when offline moves is exactly the same as for the line. In particular if the offline algorithm moves by distance x, only the matching cost is affected in the potential function and it can increase by at most $x \cdot k(h+1)/(k+1)$.

It remains to analyze the change in the potential caused by the moves of DC. In that case, we break down the DC move into *elementary moves*. Let us call *active servers* the servers adjacent to the requested point r, i.e., the ones which are moving. An elementary move ends when any server reaches either the request r or a vertex of the tree. In the latter case, another elementary move immediately follows, perhaps with a different set of active servers. We are going to prove that (5) holds for every elementary move. By summation, this implies that it holds for the entire DC move.

Consider an elementary move where q servers are moving by distance d. We need to etablish some notation first: Let M be a minimizer of $\Psi_M(X,Y)$ at the beginning of the step and A be the set of active servers. Let us imagine for now, that the requested point r is the root of the whole tree. For $a \in A$ let Q_a denote the set of DC servers in the subtree below a (but including a). 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 active servers is matched to offline server in r. We get that $\mathcal{M}(M,Y)$ increases by at most $(|A_M| - 2) \cdot d$.

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.

For D_X, it is clear that each $a \in A$, moves further away from $q_a - 1$ DC servers by distance d and gets closer to $k - q_a$ by the same distance. Thus, the change of D_X associated with a is $(q_a - 1 - (k - q_a))d = (2q_a - k - 1)d$. Overall,

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

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 $h_a - 1$ matched DC servers and gets closer to the rest $h - h_a$ of them. Thus, the change of D_M associated with a is $(2h_a - h - 1)d$, so overall we have

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

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

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

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

As the cost of DC is $q \cdot d$, we get that (5) holds, which completes the proof. □

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), also appeared in *Proc. of the 17th Symp. on Theor. Aspects of Comput. Sci.* (STACS), pages 605–613, 2000
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), also appeared in *Proc. of the 1st ACM-SIAM Symp. on Discrete Algorithms* (SODA), pp. 291–300, 1990
4. Chrobak, M., Larmore, L.L.: An optimal on-line algorithm for k-servers on trees. SIAM J. Comput. 20(1), 144–148 (1991)
5. 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)
6. Koutsoupias, E., Papadimitriou, C.H.: On the k-server conjecture. J. ACM 42(5), 971–983 (1995), also appeared in *Proc. of the 26th Symp. on Theory of Computing* (STOC), pp. 507–511, 1994
7. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for server problems. J. ACM 11(2), 208–230 (1990), also appeared as *Competitive algorithms for on-line problems* in *Proc. of the 20th Symp. on Theory of Computing* (STOC), pages 322–333, 1988
8. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM 28(2), 202–208 (1985)
9. Young, N.E.: The k-server dual and loose competitiveness for paging. Algorithmica 11(6), 525–541 (1994)

A DC for paging

The paging problem is the special case of k-server on a uniform metric. Equivalently [7], we can consider the k-server problem on a star graph, where all edges have weight $\frac{1}{2}$ and requests appear at the leaves. It known that (h,k)-paging has a deterministic competitive ratio of $\frac{k}{k-h+1}$. However, we are not aware of any explicit proof showing that DC also achieves this ratio. We give such a proof using 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.

We define

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

Where ℓ is the number of DC servers at the root.

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

Offline moves: When optimal moves any single server from one leaf to another it pays 1. The first term of the potential is not affected while the second can increase by at most one. We get that $\Delta\Phi \leq \frac{k}{k-h+1} = \frac{k}{k-h+1} \cdot OPT$.

DC moves: Let us now consider moves of DC. We distinguish between two cases depending on whether it moves one or more servers.

- $\ell > 0$: In this case, DC moves one server from the root to the requested leaf, so DC pays $1/2$. The number of servers at the root ℓ decreases by 1 and the second term decreases by 1. We get

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

  and hence $DC + \Delta\Phi = 0$.
- $\ell = 0$: In the 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 there is an OPT at the current request that was not covered when DC started moving. Thus we can upper bound $\Delta\Phi$ as:

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

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

[7] up to a fixed additive term