On the Additive Constant of the $k$-server Work Function Algorithm

Yuval Emek∗ Pierre Fraigniaud† Amos Korman‡ Adi Rosén§

Abstract

We consider the Work Function Algorithm for the $k$-server problem [2, 3]. We show that if the Work Function Algorithm is $c$-competitive, then it is also strictly $(2c)$-competitive. As a consequence of [3] this also shows that the Work Function Algorithm is strictly $(4k - 2)$-competitive.

1 Introduction

A (deterministic) online algorithm $\text{Alg}$ is said to be $c$-competitive if for all finite request sequences $\rho$, it holds that $\text{Alg}(\rho) \leq c \cdot \text{OPT}(\rho) + \beta$, where $\text{Alg}(\rho)$ and $\text{OPT}(\rho)$ are the costs incurred by $\text{Alg}$ and the optimal algorithm, respectively, on $\sigma$ and $\beta$ is a constant independent of $\rho$. When this condition holds for $\beta = 0$, then $\text{Alg}$ is said to be strictly $c$-competitive.

The $k$-server problem is one of the most extensively studied online problems (cf. [1]). To date, the best known competitive ratio for the $k$-server problem on general metric spaces is $2k - 1$ [3], which is achieved by the Work Function Algorithm [2]. A lower bound of $k$ for any metric space with at least $k + 1$ nodes is also known [4]. The question whether online algorithms are strictly competitive, and in particular if there is a strictly competitive $k$-server algorithm, is of interest for two reasons. First, as a purely theoretical question. Second, at times one attempts to build a competitive online algorithm by repeatedly applying another online algorithm as a subroutine. In that case, if the online algorithm applied as a subroutine is not strictly competitive, the resulting online algorithm may not be competitive at all due to the growth of the additive constant with the length of the request sequence.

∗Tel Aviv University, Tel Aviv, 69978 Israel. E-mail: yuvale@eng.tau.ac.il. This work was partially done during this author’s visit at LIAFA, CNRS and University Paris Diderot, supported by Action COST 295 DYNAMO.
†CNRS and University Paris Diderot, France. Email: pierre.fraigniaud@liafa.jussieu.fr. Additional support from the ANR project ALADDIN, by the INRIA project GANG, and by COST Action 295 DYNAMO.
‡CNRS and University Paris Diderot, France. Email: amos.korman@liafa.jussieu.fr. Additional support from the ANR project ALADDIN, by the INRIA project GANG, and by COST Action 295 DYNAMO.
§CNRS and University of Paris 11, France. Email: adiro@lri.fr. Research partially supported by ANR projects AlgoQP and ALADDIN.
In this paper we show that there exists a strictly competitive $k$-server algorithm for general metric spaces. In fact, we show that if the Work Function Algorithm is $c$-competitive, then it is also strictly $(2c)$-competitive. As a consequence of [3], we thus also show that the Work Function Algorithm is strictly $(4k - 2)$-competitive.

2 Preliminaries

Let $\mathcal{M} = (V, \delta)$ be a metric space. We consider instances of the $k$-server problem on $\mathcal{M}$, and when clear from the context, omit the mention of the metric space. At any given time, each server resides in some node $v \in V$. A subset $X \subseteq V$, $|X| = k$, where the servers reside is called a configuration. The distance between two configurations $X$ and $Y$, denoted by $D(X, Y)$, is defined as the weight of a minimum weight matching between $X$ and $Y$. In every round, a new request $r \in V$ is presented and should be served by ensuring that a server resides on the request $r$. The servers can move from node to node, and the movement of a server from node $x$ to node $y$ incurs a cost of $\delta(x, y)$.

Fix some initial configuration $A_0$ and some finite request sequence $\rho$. The work function $w_\rho(X)$ of the configuration $X$ with respect to $\rho$ is the optimal cost of serving $\rho$ starting in $A_0$ and ending up in configuration $X$. The collection of work function values $w_\rho(\cdot) = \{(X, w_\rho(X)) \mid X \subseteq V, |X| = k\}$ is referred to as the work vector of $\rho$ (and initial configuration $A_0$).

A move of some server from node $x$ to node $y$ in round $t$ is called forced if a request was presented at $y$ in round $t$. (An empty move, in case that $x = y$, is also considered to be forced.) An algorithm for the $k$-server problem is said to be lazy if it only makes forced moves. Given some configuration $X$, an offline algorithm for the $k$-server problem is said to be $X$-lazy if in every round other than the last round, it only makes forced moves, while in the last round, it makes a forced move and it is also allowed to move servers to nodes in $X$ from nodes not in $X$. Since unforced moves can always be postponed, it follows that $w_\rho(X)$ can be realized by an $X$-lazy (offline) algorithm for every choice of configuration $X$.

Given an initial configuration $A_0$ and a request sequence $\rho$, we denote the total cost paid by an online algorithm $\text{Alg}$ for serving $\rho$ (in an online fashion) when it starts in $A_0$ by $\text{Alg}(A_0, \rho)$. The optimal cost for serving $\rho$ starting in $A_0$ is denoted by $\text{opt}(A_0, \rho) = \min_X \{w_\rho(X)\}$. The optimal cost for serving $\rho$ starting in $A_0$ and ending in configuration $X$ is denoted by $\text{opt}(A_0, \rho, X) = w_\rho(X)$. (This seemingly redundant notation is found useful hereafter.)

Consider some metric space $\mathcal{M}$. In the context of the $k$-server problem, an algorithm $\text{Alg}$ is said to be $c$-competitive if for any initial configuration $A_0$, and any finite request sequence $\rho$, $\text{Alg}(A_0, \rho) \leq c \cdot \text{opt}(A_0, \rho) + \beta$, where $\beta$ may depend on the initial configuration $A_0$, but not on the request sequence $\rho$. $\text{Alg}$ is said to be strictly $c$-competitive if it is $c$-competitive with additive constant $\beta = 0$, that is, if for any initial configuration $A_0$ and any finite request sequence $\rho$, $\text{Alg}(A_0, \rho) \leq c \cdot \text{opt}(A_0, \rho)$. As common in other works, we assume that the online algorithm and
the optimal algorithm have the same initial configuration.

3 Strictly competitive analysis

We prove the following theorem.

**Theorem 3.1.** If the Work Function Algorithm is $c$-competitive, then it is also strictly $(2c)$-competitive.

In fact, we shall prove Theorem 3.1 for a (somewhat) larger class of $k$-server online algorithms, referred to as robust algorithms (this class will be defined soon). We say that an online algorithm for the $k$-server problem is request-sequence-oblivious, if for every initial configuration $A_0$, request sequence $\rho$, current configuration $X$, and request $r$, the action of the algorithm on $r$ after it served $\rho$ (starting in $A_0$) is fully determined by $X$, $r$, and the work vector $w_\rho(\cdot)$. In other words, a request-sequence-oblivious online algorithm can replace the explicit knowledge of $A_0$ and $\rho$ with the knowledge of $w_\rho(\cdot)$. An online algorithm is said to be robust if it is lazy, request-sequence-oblivious, and its behavior does not change if one adds to all entries of the work vector any given value $d$. We prove that if a robust algorithm is $c$-competitive, then it is also strictly $(2c)$-competitive. Theorem 3.1 follows as the work function algorithm is robust.

In what follows, we consider a robust online algorithm $\text{Alg}$ and a lazy optimal (offline) algorithm $\text{Opt}$ for the $k$-server problem. (In some cases, $\text{Opt}$ will be assumed to be $X$-lazy for some configuration $X$. This will be explicitly stated.) We also consider some underlying metric $\mathcal{M} = (V, \delta)$ that we do not explicitly specify. Suppose that $\text{Alg}$ is $\alpha$-competitive and given the initial configuration $A_0$, let $\beta = \beta(A_0)$ be the additive constant in the performance guarantee.

Subsequently, we fix some arbitrary initial configuration $A_0$ and request sequence $\rho$. We have to prove that $\text{Alg}(A_0, \rho) \leq 2\alpha \text{Opt}(A_0, \rho)$. A key ingredient in our proof is a designated request sequence $\sigma$ referred to as the anchor of $A_0$ and $\rho$. Let $\ell = \min \{ \delta(x, y) \mid x, y \in A_0, x \neq y \}$. Given that $A_0 = \{ x_1, \ldots, x_k \}$, the anchor is defined to be

$$\sigma = (x_1 \cdots x_k)^m, \text{ where } m = \left\lceil \max \left\{ \frac{2k0\text{pt}(A_0, \rho)}{\ell}, k^2, \frac{2\alpha \text{Opt}(A_0, \rho) + \beta(A_0)}{\ell} \right\} \right\rceil + 1.$$

That is, the anchor consists of $m$ cycles of requests presented at the nodes of $A_0$ in a round-robin fashion.

Informally, we shall append $\sigma$ to $\rho$ in order to ensure that both $\text{Alg}$ and $\text{Opt}$ return to the initial configuration $A_0$. This will allow us to analyze request sequences of the form $(\rho \sigma)^q$ as $q$ disjoint executions on the request sequence $\rho \sigma$, thus preventing any possibility to “hide” an additive constant in the performance guarantee of $\text{Alg}(A_0, \rho)$. Before we can analyze this phenomenon, we have to establish some preliminary properties.

**Proposition 3.2.** For every initial configuration $A_0$ and request sequence $\rho$, we have $\text{Opt}(A_0, \rho, A_0) \leq 2 \cdot \text{Opt}(A_0, \rho)$.
Proof. Consider an execution $\eta$ that (i) starts in configuration $A_0$; (ii) serves $\rho$ optimally; and (iii) moves (optimally) to configuration $A_0$ at the end of round $|\rho|$. The cost of step (iii) cannot exceed that of step (ii) as we can always retrace the moves $\eta$ did in step (ii) back to the initial configuration $A_0$. The assertion follows since $\eta$ is a candidate to realize $\text{Opt}(A_0, \rho, A_0)$. \hfill $\Box$

Since no moves are needed in order to serve the anchor $\sigma$ from configuration $A_0$, it follows that

$$\text{Opt}(A_0, \rho) \leq \text{Opt}(A_0, \rho\sigma) \leq 2 \cdot \text{Opt}(A_0, \rho).$$

Proposition 3.2 is also employed to establish the following lemma.

**Lemma 3.3.** Given some configuration $X$, consider an $X$-lazy execution $\eta$ that realizes $\text{Opt}(A_0, \rho\sigma, X)$. Then $\eta$ must be in configuration $A_0$ at the end of round $t$ for some $|\rho| \leq t < |\rho\sigma|$.

Proof. Assume by way of contradiction that $\eta$’s configuration at the end of round $t$ differs from $A_0$ for every $|\rho| \leq t < |\rho\sigma|$. The cost $\text{Opt}(A_0, \rho\sigma, X)$ paid by $\eta$ is at most $2 \cdot \text{Opt}(A_0, \rho) + D(A_0, X)$ as Proposition 3.2 guarantees that this is the total cost paid by an execution that (i) realizes $\text{Opt}(A_0, \rho, A_0)$; (ii) stays in configuration $A_0$ until (including) round $|\rho\sigma|$; and (iii) moves (optimally) to configuration $X$.

Let $Y$ be the configuration of $\eta$ at the end of round $|\rho|$. We can rewrite the total cost paid by $\eta$ as $\text{Opt}(A_0, \rho\sigma, X) = \text{Opt}(A_0, \rho, Y) + \text{Opt}(Y, \sigma, X)$. Clearly, the former term $\text{Opt}(A_0, \rho, Y)$ is not smaller than $D(A_0, Y)$ which lower bounds the cost paid by any execution that starts in configuration $A_0$ and ends in configuration $Y$. We will soon prove (under the assumption that $\eta$’s configuration at the end of round $t$ differs from $A_0$ for every $|\rho| \leq t < |\rho\sigma|$) that the latter term $\text{Opt}(Y, \sigma, X)$ is (strictly) greater than $2 \cdot \text{Opt}(A_0, \rho) + D(Y, X)$. Therefore $D(A_0, Y) + 2 \cdot \text{Opt}(A_0, \rho) + D(Y, X) < \text{Opt}(A_0, \rho, Y) + \text{Opt}(Y, \sigma, X) = \text{Opt}(A_0, \rho\sigma, X)$. The inequality $\text{Opt}(A_0, \rho\sigma, X) \leq 2 \cdot \text{Opt}(A_0, \rho) + D(A_0, X)$ then implies that $D(A_0, X) > D(A_0, Y) + D(Y, X)$, in contradiction to the triangle inequality.

It remains to prove that $\text{Opt}(Y, \sigma, X) > 2 \cdot \text{Opt}(A_0, \rho) + D(Y, X)$. For that purpose, we consider the suffix $\phi$ of $\eta$ which corresponds to the execution on the subsequence $\sigma$ ($\phi$ is an $X$-lazy execution that realizes $\text{Opt}(Y, \sigma, X)$). Clearly, $\phi$ must shift from configuration $Y$ to configuration $X$, paying cost of at least $D(Y, X)$. Moreover, since $\phi$ is $X$-lazy, and by the assumption that $\phi$ does not reside in configuration $A_0$, it follows that in each of the $m$ cycles of the round-robin, at least one server must move between two different nodes in $A_0$. (To see this, recall that each server’s move of the lazy execution ends up in a node of $A_0$. On the other hand, all $k$ servers never reside in configuration $A_0$.) Thus $\phi$ pays a cost of at least $\ell$ per cycle, and $m\ell$ altogether. A portion of this $m\ell$ cost can be charged on the shift from configuration $Y$ to configuration $X$, but we show that the remaining cost is strictly greater than $2 \cdot \text{Opt}(A_0, \rho)$, thus deriving the desired inequality $\text{Opt}(Y, \sigma, X) > 2 \cdot \text{Opt}(A_0, \rho) + D(Y, X)$.

The $k$ servers make at least $m$ moves between two different nodes in $A_0$ when $\phi$ serves the subsequence $\sigma$, hence there exists some server $s$ that makes at least $m/k$ such moves as part of
φ. The total cost paid by all other servers in φ is bounded from below by their contribution to D(Y, X). As there are k nodes in A0, at most k out of the m/k moves made by s arrive at a new node, i.e., a node which was not previously reached by s in φ. Therefore at least m/k − k moves of s cannot be charged on its shift from Y to X. It follows that the cost paid by s in φ is at least (m/k − k)ℓ plus the contribution of s to D(Y, X). The assertion now follows by the definition of m, since (m/k − k)ℓ > 2 · Opt(A0, ρ).

Since the optimal algorithm Opt is assumed to be lazy, Lemma 3.3 implies the following corollary.

Corollary 3.4. If the optimal algorithm Opt serves a request sequence of the form ρστ (for any choice of suffix τ) starting from the initial configuration A0, then at the end of round |ρσ| it must be in configuration A0.

Consider an arbitrary configuration X. We want to prove that wρσ(X) ≥ wρσ(A0) + D(A0, X). To this end, assume by way of contradiction that wρσ(X) < wρσ(A0) + D(A0, X). Fix w0 = wρσ(A0). Lemma 3.3 guarantees that an X-lazy execution η that realizes wρσ(X) = Opt(A0, ρσ, X) must be in configuration A0 at the end of some round |ρ| ≤ t < |ρσ|. Let wt be the cost paid by η up to the end of round t. The cost paid by η in order to move from A0 to X is at least D(A0, X), hence wρσ(X) ≥ wt + D(A0, X). Therefore wt < w0, which derives a contradiction, since w0 can be realized by an execution that reaches A0 at the end of round t and stays in A0 until it completes serving σ without paying any more cost. As wρσ(X) ≤ wρσ(A0) + D(A0, X), we can establish the following corollary.

Corollary 3.5. For every configuration X, we have wρσ(X) = wρσ(A0) + D(A0, X).

Recall that we have fixed the initial configuration A0 and the request sequence ρ and that σ is their anchor. We now turn to analyze the request sequence χ = (ρσ)q, where q is a sufficiently large integer that will be determined soon. Corollary 3.3 guarantees that Opt is in the initial configuration A0 at the end of round |ρσ|. By induction on i, it follows that Opt is in A0 at the end of round i · |ρσ| for every 1 ≤ i ≤ q. Therefore the total cost paid by Opt on χ is merely

\[ \text{Opt}(A0, \chi) = q \cdot \text{Opt}(A0, \rho \sigma). \] (2)

Suppose by way of contradiction that the online algorithm Alg, when invoked on the request sequence ρσ from initial configuration A0, does not end up in A0. Since Alg is lazy, we conclude that Alg is not in configuration A0 at the end of round t for any |ρ| ≤ t < |ρσ|. Therefore in each cycle of the round-robin, Alg moves at least once between two different nodes in A0, paying cost of at least ℓ. By the definition of m (the number of cycles), this sums up to Alg(A0, ρσ) ≥ mℓ ≥ 2αOpt(A0, ρ) + β(A0). By inequality (1), we conclude that Alg(A0, ρσ) > αOpt(A0, ρσ) + β(A0), in contradiction to the performance guarantee of Alg. It follows that Alg returns to the initial configuration A0 after serving the request sequence ρσ.

Consider some two request sequences τ and τ′. We say that the work vector wτ(·) is d-equivalent to the work vector wτ′(·), where d is some real, if wτ(X) − wτ′(X) = d for every X ⊆ V, |X| = k. It
is easy to verify that if $w_r(\cdot)$ is $d$-equivalent to $w_r(\cdot)$, then $w_r(\cdot)$ is $d$-equivalent to $w_{r',\cdot}(\cdot)$ for any choice of request $r \in V$. Corollary 3.5 guarantees that the work vector $w_{\rho}(\cdot)$ is $d$-equivalent to the work vector $w_{\omega}(\cdot)$ for some real $d$, where $\omega$ stands for the empty request sequence. (In fact, $d$ is exactly $w_{\rho}(A_0)$.) By induction on $j$, we show that for every prefix $\pi$ of $\rho\sigma$ and for every $1 \leq i < q$ such that $|((\rho\sigma)^i)\pi| = j$, the work vector $w_{((\rho\sigma)^i)\pi}(\cdot)$ is $d$-equivalent to the work vector $w_{\rho}(\cdot)$ for some real $d$. Therefore the behavior of the robust online algorithm $\text{Alg}$ on $\chi$ is merely a repetition ($q$ times) of its behavior on $\rho\sigma$ and

$$\text{Alg}(A_0, \chi) = q \cdot \text{Alg}(A_0, \rho) \cdot$$ (3)

We are now ready to establish the following inequality:

$$\text{Alg}(A_0, \rho) \leq \frac{\text{Alg}(A_0, \rho)}{q} \quad \text{by inequality (3)}$$

$$\leq \frac{\alpha_{\text{opt}}(A_0, \chi) + \beta(A_0)}{q} \quad \text{by the performance guarantee of Alg}$$

$$= \frac{\alpha_{\text{opt}}(A_0, \rho) + \beta(A_0)}{q} \quad \text{by inequality (2)}$$

$$\leq \frac{2\alpha_{\text{opt}}(A_0, \rho) + \beta(A_0)}{q} \quad \text{by inequality (1)}$$

For any real $\epsilon > 0$, we can fix $q = \lceil \beta(A_0)/\epsilon \rceil + 1$ and conclude that $\text{Alg}(A_0, \rho) < 2\alpha_{\text{opt}}(A_0, \rho) + \epsilon$. Theorem 3.1 follows.

As the Work Function Algorithm is known to be $(2k-1)$-competitive [3], we also get the following corollary.

**Corollary 3.6.** The Work Function Algorithm is strictly $(4k-2)$-competitive.

**Acknowledgments** We thank Elias Koutsoupias for useful discussions.

**References**

[1] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[2] M. Chrobak and L.L. Larmore. The server problem and on-line games. In *On-line algorithms: Proc. of a DIMACS Workshop*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 7, pages 11–64, 1991.
[3] E. Koutsoupias and C.H. Papadimitriou. On the $k$-server conjecture. *J. ACM*, 42(5):971–983, 1995.

[4] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.