On the Competitiveness of a Modified Work Function Algorithm for Solving the On-Line \( k \)-Server Problem

Tomislav Rudec  
Faculty of Electrical Engineering, University of Osijek  
Kneza Trpimira 2b, 31000 Osijek, Croatia  
Tomislav.Rudec@etfos.hr

Robert Manger  
Department of Mathematics, University of Zagreb  
Bijenička cesta 30, 10000 Zagreb, Croatia  
Robert.Manger@math.hr

Abstract. We study a modification of the well known work function algorithm (WFA) for solving the on-line \( k \)-server problem. Our modified WFA is based on a moving window, i.e. on the approximate work function that takes into account only a fixed number of most recent on-line requests. In this paper we prove that, in contrast to the original WFA, the modified WFA is not competitive. Our proof is based on a suitably constructed problem instance showing that the performance of the modified WFA can be an arbitrary number of times worse than optimal.

Keywords. on-line problems, on-line algorithms, \( k \)-server problem, work function algorithm (WFA), moving windows, competitiveness.

1. Introduction

This paper deals with the \( k \)-server problem [9], which belongs to a broader family of on-line problems [7]. In the \( k \)-server problem one has to decide how \( k \) mobile servers should serve a sequence of on-line requests. To solve the \( k \)-server problem, one needs a suitable on-line algorithm [7]. The goal of such an algorithm is not only to serve requests as they arrive, but also to minimize the total cost of serving. A desirable property of an on-line algorithm is its competitiveness [10]. Vaguely speaking, an algorithm is competitive if its performance is only a bounded number of times worse than optimal.

Many on-line algorithms for solving the \( k \)-server problem can be found in literature. Among them, the best characteristics regarding competitiveness exhibits the so-called work function algorithm (WFA) [2,8]. In spite of its theoretical importance, the WFA is seldomly used in practice due to its prohibitive computational complexity.

In a recent paper [3] a simple modification of the WFA has been proposed, which is based on a moving window. Such modified “lightweight” WFA is more suitable for practical purposes since its computational complexity can be controlled by the window size. The paper [3] demonstrates experimentally that the performance of the WFA in terms of the incurred total cost is not degraded by the introduced modification. Namely, with a reasonably large window the modified WFA usually achieves the same performance as the original WFA.

The aim of this paper is to investigate competitiveness of the modified WFA from [3]. More precisely, this paper aims to prove that the modified WFA is in fact not competitive. The proof will be based on a suitably constructed problem instance, where the ratio between the cost incurred by the algorithm and the optimal cost can be arbitrary large.

The paper is organized as follows. Section 2 and 3 list some preliminaries about the \( k \)-server problem, the corresponding algorithms

Proceedings of the ITI 2008 30th Int. Conf. on Information Technology Interfaces, June 23-26, 2008, Cavtat, Croatia
including the original WFA, and competitiveness. Section 4 presents the modified WFA and describes its properties. Section 5 is the main part of the paper - it contains the proof that the modified WFA is not competitive. The final Section 6 gives a conclusion.

2. The $k$-server problem

In the $k$-server problem [9] we have $k$ servers each of which occupies a location (point) in a fixed metric space $M$. Repeatedly, a request $r_i$ at some location $x \in M$ appears. Each request must be served by a server before the next request arrives. To serve a new request at $x$, an on-line algorithm must move a server to $x$ unless it already has a server at that location. The decision which server to move may be based only on the already seen requests $r_1, r_2, \ldots, r_{i-1}, r_i$, thus it must be taken without any information about the future requests $r_{i+1}, r_{i+2}, \ldots$. Whenever the algorithm moves a server from a location $x$ to a location $y$, it incurs a cost equal to the distance between $x$ and $y$ in $M$. The goal is not only to serve requests, but also to minimize the total distance moved by all servers.

As a concrete instance of the $k$-server problem, let us consider the set $M$ of $m = 5$ Croatian cities shown in Figure 1 with distances given. Suppose that $k = 3$ different hail-defending rocket systems are initially located at Osijek, Zagreb and Split. If the next hail alarm appears for instance in Karlovac, then our hail-defending on-line algorithm has to decide which of the tree rocket systems should be moved to Karlovac. Seemingly the cheapest solution would be to move the nearest system from Zagreb. But such a choice could be wrong if, for instance, all forthcoming requests would appear in Zagreb, Karlovac and Osijek and none in Split.

3. Algorithms and competitiveness

The simplest on-line algorithm for solving the $k$-server problem is the greedy algorithm (GREEDY) [7]. It serves the current request in the cheapest possible way, by ignoring history altogether. Thus GREEDY sends the nearest server to the requested location.

A slightly more sophisticated solution is the balanced algorithm (BALANCE) [9], which attempts to keep the total distance moved by various servers roughly equal. Consequently, BALANCE employs the server whose cumulative distance traveled so far plus the distance to the requested location is minimal.

The most celebrated solution to the $k$-server problem is the work function algorithm (WFA) [2,8]. To serve the request $r_i$, the WFA switches from the current server configuration $S^{(i-1)}$ to a new configuration $S^{(i)}$, obtained from $S^{(i-1)}$ by moving one server into the requested location (if necessary). Among $k$ possibilities (any of $k$ servers could be moved) $S^{(i)}$ is chosen so that it minimizes the so-called work function $F()$. More precisely, $S^{(i)}$ is chosen so that

$$
F(S^{(i)}) = C_{OPT}(S^{(0)}, r_1, r_2, \ldots, r_i, S^{(i)}) + d(S^{(i-1)}, S^{(i)})
$$

becomes minimal. As we see, $F(S^{(i)})$ is defined here as a sum of two parts.

- The first part is the minimum total cost of starting from $S^{(0)}$, serving in turn $r_1, r_2, \ldots, r_i$, and ending up in $S^{(i)}$. 

780
• The second part is the distance traveled by a server to switch from $S^{(i-1)}$ to $S^{(i)}$.

Note that an on-line algorithm ALG can only approximate the performance of the corresponding optimal off-line algorithm OPT, which knows the whole input in advance and deals with input data as they arrive at minimum total cost. Such desirable approximation property of ALG is formally described by the notion of competitiveness. ALG is said to be competitive if its performance is estimated to be only a bounded number of times worse than that of OPT on any input. More precisely [10], let $\sigma = (r_1, r_2, \ldots, r_n)$ be a sequence of requests. Denote with $C_{\text{ALG}}(\sigma)$ the total cost incurred by ALG on $\sigma$, and with $C_{\text{OPT}}(\sigma)$ the minimum total cost on $\sigma$. For a chosen constant $p$, we say that ALG is $p$-competitive if there exists another constant $q$ such that on every $\sigma$ it holds:

$$C_{\text{ALG}}(\sigma) \leq p \cdot C_{\text{OPT}}(\sigma) + q.$$ 

There are many interesting results dealing with competitiveness. For instance, it can be proved [9] that any hypothetical $p$-competitive algorithm for the $k$-server problem must have $p \geq k$. Also, it is easy to check [7] that both GREEDY and BALANCE are not competitive, i.e. they have no bounded $p$. Finally, it has been proved in [9] that the WFA is $(2k-1)$-competitive.

The WFA can be regarded as the “most competitive” algorithm for the $k$-server problem. Namely, its estimated value of $p$ is several orders of magnitude lower than for any other known algorithm of the same type [1]. It is widely believed that the WFA is in fact $k$-competitive (thus achieving the best possible $p$), but this hypothesis has not been proved except for some special cases [2].

4. The modified WFA

The modification of the WFA from [3], denoted as the $w$-WFA, is based on the idea that the sequence of requests and configurations should be examined through a moving window of size $w$. More precisely, in its $i$-th step the $w$-WFA acts as if $r_{i-w+1}, r_{i-w+2}, \ldots, r_{i-1}, r_i$ was the whole sequence of previous requests, and as if $S^{(i-w)}$ was the initial configuration of servers. In other words, the work function $F()$, originally defined by (1), is redefined in the following way:

$$F(S^{(i)}) = C_{\text{OPT}}(S^{(i-w)}, r_{i-w+1}, r_{i-w+2}, \ldots, r_{i-1}, r_i, S^{(i)}) + d(S^{(i-1)}, S^{(i)}).$$

(2)

The main advantage of the $w$-WFA compared to the original WFA is much smaller and controllable computational complexity. As it has been shown in [4], both versions of the WFA can be implemented relatively efficiently by network flow techniques [5], so that one step of the algorithm reduces to $k$ suitably constructed minimum-cost maximal flow problems. However, the $i$-th step of the original WFA requires $O(k^2 \cdot (i+k)^2)$ operations, while the $w$-WFA needs $O(k^2 \cdot (w+k)^2)$ operations per step. Thus the complexity of the original WFA rises with each step, while the complexity of the $w$-WFA remains constant and depends on the chosen $w$.

The paper [3] presents some experimental results measuring the performance of the $w$-WFA in terms of the incurred total cost. Several instances of the $k$-server problem have been repeatedly solved by the $w$-WFA with different values of $w$. The same instances have also been solved by the other mentioned algorithms. Comparison of results indicates that the $w$-WFA always achieves the same performance as the original WFA if $w$ is large enough. More precisely, the equivalence between the $w$-WFA and the original WFA on a sequence of 100 requests is reached with $w$ between 16 and 30. Also, the $w$-WFA performs better than BALANCE or GREEDY if the sequence of requests has non-uniform distribution.

5. Proof of non-competitiveness

In this section we will show that for a chosen $w$ the $w$-WFA is not competitive. Since the 1-WFA is equivalent to GREEDY, and GREEDY in known to be non-competitive, we can restrict to $w \geq 2$. 

781
Also, we can take $k \geq 2$ since otherwise the algorithm has nothing to decide.

For a chosen $w \geq 2$ let us construct a metric space $M$ consisting of $k+1$ locations. The first two locations $x_1$ and $x_2$ constitute a “distant island”. The remaining $k-1$ locations $y_1$, $y_2$, ..., $y_{k-1}$ constitute the “mainland”. Let the distance between $x_1$ and $x_2$ be $\delta$. Assume that for all $i$ and $j$ the distance between $x_i$ and $y_j$ is $\geq \Delta$, where $\Delta > w\delta$. Assume also that the distance between $x_1$ and $y_1$ is exactly $\Delta$. The situation is illustrated by Figure 2.

![Figure 2. A suitably constructed metric space.](image)

The initial configuration of servers is such that one server resides at $x_2$ and the remaining servers occupy $y_1$, $y_2$, ..., $y_{k-1}$, respectively. The sequence of requests $r_1$, $r_2$, ..., $r_n$, is given so that $r_1$ appears at $x_1$, $r_2$ at $x_2$, $r_3$ again at $x_1$, $r_4$ again at $x_2$, ..., etc. Thus the requests alternate at the island locations $x_1$ and $x_2$ and they never occur on the mainland.

We claim that in the described situation the $w$-WFA serves requests in a “ping-pong” fashion, by moving in each step the server already residing on the island, and never employing any of the remaining servers from the mainland. Thus the service offered by the $w$-WFA looks as described in Table 1.

Now we explain in more detail why the $w$-WFA is forced to serve requests by using only one server. Let us analyze the performance of the algorithm in the $i$-th step when the request $r_i$ appears, provided that all previous requests $r_1$, $r_2$, ..., $r_{i-1}$ have already been served in the ping-pong fashion. We consider the following three cases: $i = 1, 2 \leq i \leq w$, and $i > w$, respectively.

**Case 1: $i = 1$.** The first request $r_1$ appears at the location $x_1$. The algorithm has to decide how to serve $r_1$: either by moving the server from $x_2$ to $x_1$, or by transporting any of the mainland servers to $x_2$. For $i = 1$ both parts of the work function $F()$ from (1) reduce to the distance that the chosen server has to cross. Thus choosing the server that minimizes $F()$ means choosing the nearest server, and since $\delta < \Delta$ this is certainly the one already on the island. Thus the ping-pong begins.

**Case 2: $2 \leq i \leq w$.** For such value of $i$ the work function $F()$ has its full standard form (1). Since the previous requests have been served in the ping-pong fashion, we are sure that the current server configuration $S^{(i-1)}$ involves only one server on the island. The algorithm has to choose a new configuration $S^{(i)}$ so that the value $F(S^{(i)})$ from (1) is minimal. Let $\tilde{S}^{(i)}$ be the configuration obtained from $S^{(i-1)}$ by resuming the ping-pong, i.e. by moving again the server on the island. Since $\tilde{S}^{(i)}$ is gradually obtained from $S^{(0)}$ through a series of $i$ ping-pong moves and servings at total cost $i\delta$, it follows that specially

$$C_{\text{OPT}}(S^{(0)}, r_1, r_2, \ldots, r_i, \tilde{S}^{(i)}) \leq i\delta \leq w\delta.$$  

On the other hand, obviously,

$$d(S^{(i-1)}, \tilde{S}^{(i)}) = \delta.$$  

<table>
<thead>
<tr>
<th>request</th>
<th>location</th>
<th>server</th>
<th>cost of move serving</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>$x_1$</td>
<td>$x_2 \rightarrow x_1$</td>
<td>$\delta$</td>
</tr>
<tr>
<td>$r_2$</td>
<td>$x_2$</td>
<td>$x_1 \rightarrow x_2$</td>
<td>$\delta$</td>
</tr>
<tr>
<td>$r_3$</td>
<td>$x_1$</td>
<td>$x_2 \rightarrow x_1$</td>
<td>$\delta$</td>
</tr>
<tr>
<td>$r_4$</td>
<td>$x_2$</td>
<td>$x_1 \rightarrow x_2$</td>
<td>$\delta$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
By combining the above two relations, we get
\[ F(\hat{S}(i)) \leq (w + 1)\delta < 2\Delta. \]

Let us now consider a configuration \(\hat{S}(i)\) obtained from \(S^{(i-1)}\) by bringing a new server from the mainland to the island. It obviously holds that:
\[ d(S^{(i-1)}, \hat{S}(i)) \geq \Delta. \]

On the other hand, since \(\hat{S}(i)\) requires two servers on the island, the optimal way of serving \(r_1, r_2, \ldots, r_i\) starting from \(S(0)\) and ending up in \(\hat{S}(i)\) should also at some stage include pulling a server from the mainland. Consequently,
\[ C_{\text{OPT}}(S(0), r_1, r_2, \ldots, r_i, \hat{S}(i)) \geq \Delta. \]

By putting the last two inequalities together, we get
\[ F(\hat{S}(i)) \geq 2\Delta. \]

So we see that \(F(\hat{S}(i))\) is strictly larger than \(F(S(i))\). Therefore the algorithm can never use a configuration of the form \(\hat{S}(i)\), and the only thing it can do is to choose \(S(i)\). Thus indeed, the ping-pong continues.

Case 3: \(i > w\). For such value of \(i\) the work function \(F()\) takes its redefined form shown by (2). The algorithm does not remember any more the whole history; it acts as if \(S^{(i-w)}\) was the initial configuration and as if \(r_{i-w+1}\) was the first request. Since all previous requests have been served in the ping-pong fashion, it follows that the “initial” configuration \(S^{(i-w)}\) again involves only one server on the island (however, we cannot be sure if it resides at \(x_1\) or \(x_2\)). The same property also holds for the current configuration \(S^{(i-1)}\). The algorithm has to choose a new configuration \(S(i)\) so that the value of \(F(S(i))\) from (2) is minimal. Let again \(\hat{S}(i)\) be the configuration obtained from \(S^{(i-1)}\) by resuming the ping-pong, and let \(\hat{S}(i)\) be any configuration obtained from \(S^{(i-1)}\) by bringing a new server from the mainland to the island. By very similar reasoning as in the previous case, we can show that \(F(\hat{S}(i))\) is < \(2\Delta\), while \(F(S(i))\) must be \(\geq 2\Delta\). Thus the algorithm is again forced to choose \(\hat{S}(i)\), and the ping-pong resumes even further.

After we have established that the \(w\)-WFA really serves requests strictly in the ping-pong fashion as specified by Table 1, it is easy to calculate its total cost of serving. Indeed, for a request sequence of length \(n\), the total cost amounts to \(\delta n\), thus it increases linearly with \(n\).

Next, we have to analyze performance of the optimal algorithm \(\text{OPT}\) on our data. For a sufficiently large \(n\), \(\text{OPT}\) serves the sequence \(r_1, r_2, \ldots, r_n\) in the following way: it transports immediately in the first step an additional server from the mainland to the island. More precisely, \(\text{OPT}\) moves the server from \(y_1\) to \(x_1\) at cost \(\Delta\). After that, \(\text{OPT}\) can serve the whole sequence of requests with no additional server movements or costs. The initial effort of bringing the server from the mainland to the island will pay off as soon as \(n > \Delta/\delta > w\). The \(w\)-WFA is not able to recognize the optimal solution since it “forgets” some history and acts as if the sequence length \(n\) was always \(\leq w\).

Finally, let us combine all our estimates and complete the proof. Indeed, if the \(w\)-WFA was \(p\)-competitive for some (finite) \(p\), then the ratio between its cost and the cost of \(\text{OPT}\) would be bounded and converging to \(p\) as \(n\) rises. However, we see that the cost ratio is equal to \((\delta n)/\Delta\), i.e. it can be arbitrarily large if \(n\) is large enough. Thus the \(w\)-WFA cannot be competitive.

6. Conclusion

The considered modification of the WFA is based on a moving window with a fixed size. On the other hand, the original WFA can be interpreted as an extreme case of the modified WFA where the window size is infinite. In this paper we have shown that, in contrast to the original WFA, the modified WFA is never competitive, no matter how large is the window that has been chosen. Thus, by limiting the original infinite window to some finite size, we certainly improve the algorithm speed, but we simultaneously lose the interesting and important property of competitiveness.
The presented result may look rather disappointing, but in fact it has been expected. The result is seemingly in contradiction with the recorded experimental evidence showing that in reality the modified WFA closely mimics the performance of the original WFA. However, one must take into account that competitiveness is a very rigid and demanding theoretical criterion, which tends to disqualify many otherwise good algorithms.

The problem instance used in our proof is of course extremely artificial. We believe that on realistic problems and with a reasonably large window the difference between the modified and the original WFA could not be spotted. Thus according to our belief, the modified WFA still captures the advantages of its competitive original and performs better than simple heuristics.

At this moment, competitiveness is the only widely accepted mechanism for assessing “goodness” of on-line algorithms. It would be advantageous if some other mechanisms could be devised, which would allow more accurate ranking of non-competitive algorithms.

References