Computing and Informatics, Vol. 29, 2010, 681–700

THE DESIGN AND ANALYSIS OF A MODIFIED WORK FUNCTION ALGORITHM FOR SOLVING THE ON-LINE K-SERVER PROBLEM

Alfonzo Baumgartner, Tomislav Rudec

Faculty of Electrical Engineering, University of Osijek
Kneza Trpimira 2b, 31000 Osijek, Croatia
e-mail: baumgart@etfos.hr, tomo@ffos.hr

Robert Manger

Department of Mathematics, University of Zagreb
Bijenička cesta 30, 10000 Zagreb, Croatia
e-mail: manger@math.hr

Manuscript received 2 February 2009; revised 27 May 2009
Communicated by Ralf Klasing

Abstract. In this paper we study a modified work function algorithm (WFA) for solving the on-line k-server problem. Our modification is based on a moving window, i.e. on an approximate work function that takes into account only a fixed number of most recent on-line requests. We give a precise specification of the modified WFA, investigate its competitiveness, and explain how it can be implemented efficiently by network flows. We also present experiments that measure the performance and computational complexity of the implemented algorithm. The results of the paper can be summarized as follows: the modified WFA is not competitive, but according to the experiments it still provides almost the same quality of serving as the original WFA while running much faster.

Keywords: On-line problems, on-line algorithms, k-server problem, work function algorithm (WFA), moving windows, competitiveness, implementation, network flows, experiments, performance, computational complexity

Mathematics Subject Classification 2000: 05C85, 68Q25, 90B10, 90B35, 90C27, 90C35

1 INTRODUCTION

This paper deals with the *k-server problem* [10], which belongs to a broader family of *on-line problems* [7], and can also be regarded as a form of scheduling. 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* [12]. Vaguely speaking, an algorithm is competitive if its performance is only a bounded number of times worse than optimal.

There are various on-line algorithms for solving the k-server problem found in literature. Among them, the best characteristics regarding competitiveness are exhibited by the *work function algorithm* (WFA) [2, 9]. In spite of its importance and interesting properties, the WFA is impractical 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 seems to be more suitable for practical purposes, since its computational complexity as well as its quality of serving can hopefully be controlled by the window size. Namely, with a small window the modified WFA is expected to run very quickly, while with a reasonably large window it might achieve the same performance in terms of the incurred total cost as the original WFA.

The aim of this paper is to provide a rigorous evaluation of the ideas from [3]. Thus the paper investigates various aspects of the proposed modified WFA, including its competitiveness, efficient implementation, performance, and computational complexity.

The paper is organized as follows. Section 2 lists all necessary preliminaries about the k-server problem, the corresponding algorithms, the original WFA, and competitiveness. Section 3 gives a precise specification of the modified WFA, and explains the motivation for introducing such an algorithm. Section 4 contains a proof that, in contrast to the original WFA, the modified WFA is not competitive. Section 5 describes an efficient implementation of the modified WFA, which is based on reducing each step of the algorithm to a set of network flow problems. Section 6 reports on a series of experiments, where both the performance and computational complexity of the implemented algorithm have been measured and compared vs. the original WFA and some other algorithms. The final Section 7 gives a conclusion.

2 PRELIMINARIES

In the *k-server problem* [10] we have k servers each of which occupies a location (point) in a fixed metric space M consisting of m locations. 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, ..., r_{i-1}, r_i, thus it must be taken without any information about the future requests r_{i+1}, r_{i+2}, ... 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 three 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.

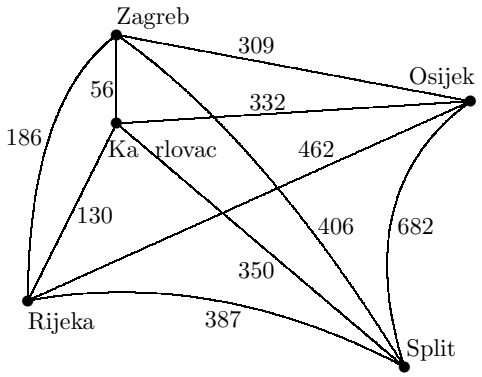


Fig. 1. A k-server problem instance

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) [10], 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, 9]. 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

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

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

- The first part, usually called the *work function*, is the minimum total cost of starting from $S^{(0)}$, serving in turn $r_1, r_2, \ldots, r_i$, and ending up in $S^{(i)}$.
- 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 [12], let $\sigma = (r_1, r_2, \ldots, r_n)$ be a sequence of requests. Denote with $C_{\mathrm{ALG}}(\sigma)$ the total cost incurred by ALG on σ, and with $C_{\mathrm{OPT}}(\sigma)$ the minimum total cost on σ. For a chosen constant α, we say that ALG is α-*competitive* if there exists another constant β such that on every σ it holds:

$$C_{\mathrm{ALG}}(\sigma) \leq \alpha \cdot C_{\mathrm{OPT}}(\sigma) + \beta.$$

There are many interesting results dealing with competitiveness. For instance, it can be proved [10] that any hypothetical α-competitive algorithm for the k-server problem must have $\alpha \geq k$. Also, it is easy to check [7] that both GREEDY and BALANCE are not competitive, i.e. they have no bounded α. 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 α 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 α), but this hypothesis has not been proved except for some special cases [2].

3 THE MODIFIED WFA

As can be seen from (1), the original WFA is extremely complex and therefore not suitable for practical purposes. Namely, each of its steps involves k demanding optimization problem instances plus some additional arithmetics. Even worse, the complexity of the i^{th} step grows with i since each of the involved optima depends on the whole list of requests r_1, r_2, $\ldots$, r_{i-1}, r_i. Consequently, the original WFA gradually slows down until it becomes intolerably slow.

The modification of the WFA from [3] has been proposed with the intention to overcome the described drawbacks of the original algorithm. The modified WFA, denoted more precisely as the w-WFA, is based on the idea that the sequence of previous requests and configurations should be examined through a moving window of size w. 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 objective function $F()$, originally defined by (1), is redefined in the following way:

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

The main advantage of the w-WFA compared to the original WFA is much lower and controllable computational complexity. Namely, each step of the w-WFA has roughly the same complexity, and the algorithm does not slow down any more. Moreover, the complexity of one step can be controlled by the window size w, i.e. smaller w means faster response. On the other hand, one can hope that with a sufficiently large w the w-WFA would still approximate the behavior of the original WFA.

4 PROOF OF NON-COMPETITIVENESS

Now we will show that for a chosen w the w-WFA is not competitive, in spite of its similarity to the original WFA. Since the 1-WFA is equivalent to GREEDY, and GREEDY is known to be non-competitive, we can restrict to $w \geq 2$. Also, we can take $k \geq 2$ since otherwise the algorithm has nothing to decide.

For a chosen $w \geq 2$ and $k \geq 2$ 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 δ. 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 Δ. The situation is illustrated by Figure 2.

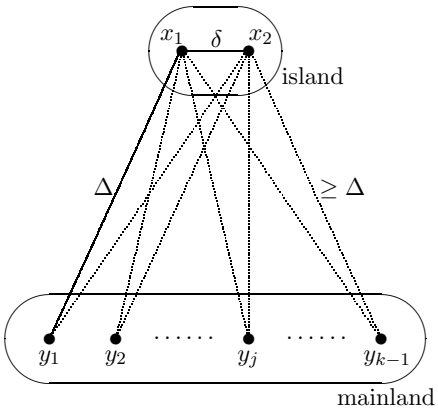


Fig. 2. A suitably constructed metric space

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.

request	location	server move	cost of serving
r_1	x_1	$x_2 \to x_1$	δ
r_2	x_2	$x_1 \to x_2$	δ
r_3	x_1	$x_2 \to x_1$	δ
r_4	x_2	$x_1 \to x_2$	δ
...	...	...	...

Table 1. Ping-pong serving

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: $i = 1$, $2 \le i \le w$, 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_1. For $i = 1$ both parts of the objective 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: $2 \le i \le w$. For such value of i the objective 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 $\bar{S}^{(i)}$ be the configuration obtained from $S^{(i-1)}$ by resuming the ping-pong, i.e. by moving again the server on the island. Since $\bar{S}^{(i)}$ 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

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

On the other hand, obviously,

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

By combining the above two relations, we get

$$F(\bar{S}^{(i)}) \leq (w+1)\delta < 2\Delta.$$

Let us now consider a configuration $\widehat{S}^{(i)}$ obtained from $S^{(i-1)}$ by bringing a new server from the mainland to the island. It obviously holds that:

$$d(S^{(i-1)}, \widehat{S}^{(i)}) \geq \Delta.$$

On the other hand, since $\widehat{S}^{(i)}$ requires two servers on the island, the optimal way of serving r_1, r_2, ..., r_i starting from $S^{(0)}$ and ending up in $\widehat{S}^{(i)}$ should also at some stage include pulling a server from the mainland. Consequently,

$$C_{\mathrm{OPT}}(S^{(0)}, r_1, r_2, \ldots, r_i, \widehat{S}^{(i)}) \geq \Delta.$$

By putting the last two inequalities together, we get

$$F(\widehat{S}^{(i)}) \geq 2\Delta.$$

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

Case 3: $i > w$. For such value of i the objective function $F()$ takes its redefined form shown by (2). The algorithm does not remember the whole history any more; 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 $\bar{S}^{(i)}$ be the configuration obtained from $S^{(i-1)}$ by resuming the ping-pong, and let $\widehat{S}^{(i)}$ be any configuration obtained from $S^{(i-1)}$ by bringing a new server from the mainland to the island. By similar reasoning as in the previous case, we can show that $F(\bar{S}^{(i)})$ is $< 2\Delta$, while $F(\widehat{S}^{(i)})$ must be $\geq 2\Delta$. Thus the algorithm is again forced to choose $\bar{S}^{(i)}$, 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 δn, thus it increases linearly with n.

Next, we have to analyze performance of the optimal algorithm OPT on our data. For a sufficiently large n, 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, OPT moves the server from y_1 to x_1 at cost Δ. After that, 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 α-competitive for some (finite) α, then the ratio between its cost and the cost of OPT would be bounded and converging to α 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.

5 IMPLEMENTATION BY NETWORK FLOWS

In order to implement the w-WFA, we start from the fact that the optimal off-line algorithm OPT can be implemented relatively easily by network flow techniques [4]. Namely, according to [5], finding the optimal strategy to serve a sequence of requests $r_1, r_2, \ldots, r_n$ by k servers can be reduced to computing the minimal-cost maximal flow on a suitably constructed network with $2n + k + 2$ nodes. The details of this construction are shown in Figure 3.

As we can see from Figure 3, the network corresponding to the off-line problem consists of a source node, a sink node, and three additional layers of nodes. The first layer represents the initial server configuration $S^{(0)}$, i.e. node $s_j^{(0)}$ corresponds to the initial location of the j^{th} server. The remaining two layers represent the whole sequence of requests, i.e. nodes r_p and r'_p both correspond to the location of the p^{th} request.

Some pairs of nodes are connected by arcs, as shown in Figure 3. Note that an r_p is connected only to the associated r'_p. Also, a link between an r'_p and an r_q exists only if $q > p$. All arcs are assumed to have unit capacity. The costs of arcs leaving the source or entering the sink are 0. An arc connecting r_p with r'_p has the cost $-L$, where L is a suitably chosen very large positive number. All other arc costs are equal to *distances* between corresponding locations.

It is obvious that the maximal flow through the network shown in Figure 3 must have the value k. Moreover, the maximal flow can be decomposed into k disjunct unit flows from the source to the sink. Each unit flow determines the trajectory of the corresponding server and the requests that are accomplished by that server. If the chosen constant L is large enough, then the minimal-cost maximal flow will be forced to use all arcs between r_p and r'_p, thus assuring that all requests will be served at minimum cost.

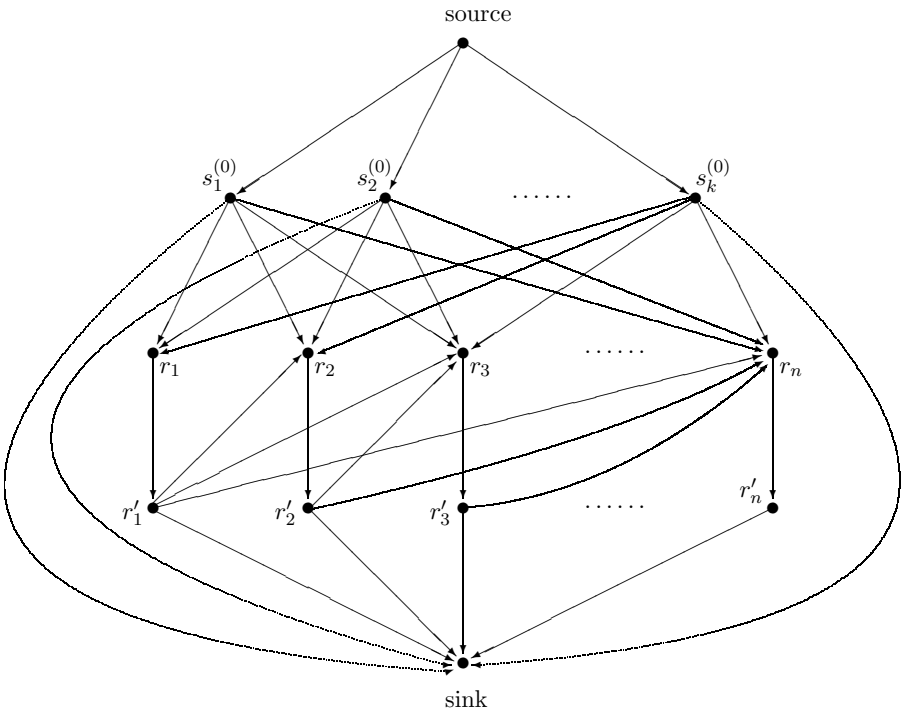


Fig. 3. Finding the optimal solution to the off-line k-server problem

According to Definition (1), the i^{th} step of the WFA consists of k optimization problem instances, plus some simple arithmetics. So there is a possibility to implement the WFA by using the above mentioned network flow techniques. It is true, however, that the optimization problems within the WFA are not quite equivalent to off-line problems, namely there is an additional constraint regarding the final configuration of servers. Still, the construction from [5] can be used after a slight modification. More precisely, the i^{th} step of the WFA can be reduced to k minimal-cost maximal flow problems, each on a network with $2i + 2k$ nodes. One of the involved networks is shown in Figure 4. Note that the network size rises with i.

As we can see from Figure 4, one of k networks used to implement the i^{th} step of the WFA is very similar to the previously described network used to find the optimal solution of the off-line problem. The main difference is that the fourth layer of nodes has been added, which is analogous to the first layer, and which specifies the currently chosen version of the final server configuration $S^{(i)}$. Note that the second and third layers now correspond only to requests $r_1, r_2, \ldots, r_{i-1}$. Still, since the final configuration $S^{(i)}$ always covers the location of the last request r_i, we are

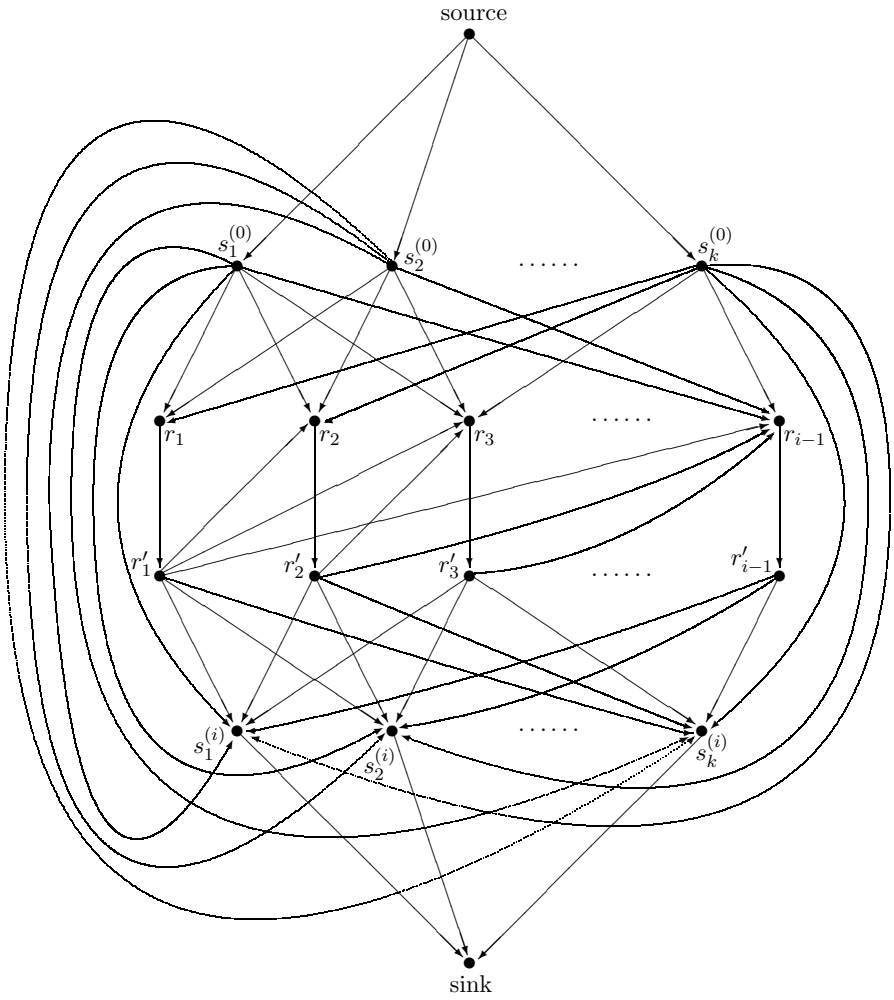


Fig. 4. Solving one of k optimization problems within the i^{th} step of the WFA

sure that r_i will also be served with no additional cost. When we switch from one particular version of $S^{(i)}$ to another, the structure of the whole network remains the same, only the costs of arcs entering the fourth level must be adjusted in order to reflect different final setting of servers.

As it has been explained in Section 3, the w-WFA is only a modified version of the WFA, where the sequence of previous requests and configurations is examined through a moving window of size w. Obviously, the w-WFA can be implemented by network flow techniques in exactly the same way as the original WFA. More precisely,

the i^{th} step of the w-WFA can again be reduced to k minimal-cost maximal flow problems, but now each of those problems is posed on a network built with $2w + 2k$ nodes, as shown in Figure 4. Note that the network size now does not change any more with i.

To complete the proposed implementation of the w-WFA, it is necessary to incorporate a suitable procedure for finding network flows. Our chosen procedure for solving minimal-cost maximal flow problems follows the well known generic *flow augmentation* method [4] with some adjustments. Thus we start with a flow that is not of maximal value but has the minimal cost among those with that value. Then in each iteration we augment the value of the current flow in such a way that it still has the minimal cost among those with the same value. After a sufficient number of iterations we obtain the minimal-cost maximal flow.

In our particular case the procedure can be started with the null flow. Namely, since the involved networks are acyclic, the null flow obviously has the minimal cost among those with value 0. In each iteration, augmentation is achieved by finding a shortest path in the corresponding *displacement network* [4]. Since the maximal flow has value k and each augmentation increments the flow value by one unit, finding the minimal-cost maximal flow reduces to exactly k single-source shortest path problems.

The last detail within our implementation of the w-WFA is the choice of an appropriate algorithm for finding the shortest paths. It is well known that the fastest among such algorithms is the one by Dijkstra [8]. However, Dijkstra's algorithm can be applied only to networks whose arc costs are nonnegative. At first sight, our networks do not qualify since they contain negative costs $-L$. Still, it turns out that Dijkstra's algorithm can be used after a suitable transformation of arc costs, as shown in [6].

In the remaining part of this section let us give some rough estimates of the computational complexity. As we have just described, our implementation of the original WFA is based on reducing the i^{th} step of the WFA to k minimal-cost maximal flow problems, each on a network with $2i+2k$ nodes. Any of those minimal-cost maximal flow problems is further reduced to k single-source shortest path problems on networks with the same size. All path problems are finally solved by Dijkstra's algorithm. The w-WFA is implemented in the same way, except that the networks involved in the i^{th} step have size $2w + 2k$.

It is well known that Dijkstra's algorithm has at most a quadratic running time. Since the i^{th} step of the WFA consists of k^2 applications of Dijkstra, and all those applications are on networks of size $2i+2k$, it follows that the i^{th} step of the WFA has computational complexity $\mathcal{O}(k^2 \cdot (i + k)^2)$. Similarly, the computational complexity of the w-WFA is $\mathcal{O}(k^2 \cdot (w + k)^2)$ per step.

The above estimates are in accordance with our expectations. Indeed, the complexity of the i^{th} step of the w-WFA does not rise with i as for the original WFA, but it still exhibits a nonlinear dependency on k and w. Consequently, the w-WFA is faster than the original WFA, but it is still rather complex compared to simple heuristics such as GREEDY or BALANCE, whose steps can easily be implemented

in $\mathcal{O}(k)$ operations. Note that we deal here with worst-case estimates, which take into account only input size, while ignoring actual input values such as actual distances among requested locations.

6 EXPERIMENTS AND THEIR RESULTS

In order to obtain experimental results, we have developed a C++ program that implements the w-WFA for any given w. Implementation is based on network flows, as described in the previous section. To allow comparison, we have also realized some other on-line algorithms, such as GREEDY, BALANCE and the original WFA. In addition, we have made a program that implements the optimal off-line algorithm OPT. The source code of all programs is publicly available and can be downloaded from the web address `http://www.math.hr/~manger/ModifiedWFA.zip`.

All programs were executed on a Linux cluster whose each node consists of two 2.8 GHz CPU-s with 2 GB of memory. Only one cluster node was employed to run one program. Thanks to using the MPI package [11], our programs were able to distribute their workload among both CPU-s. Such limited form of parallelism resulted in speeding-up all algorithms approximately by factor of two. Still, relative speeds of different algorithms remained roughly the same as for sequential computing.

In our experiments we tried to explore relative strengths and weaknesses of the considered algorithms in a systematic way. We used problem instances that differ in 4 parameters: the total number of locations m, the number of servers k, the number of consecutive requests n, and the type of distribution of requests among locations. For each parameter we tried two clearly distinct values. In this way we obtained altogether $2 \times 2 \times 2 \times 2 = 16$ distinct problem instances.

The number of locations has been chosen as $m = 25$ or $m = 15\,112$, the number of servers as $k = 3$ or $k = 10$, while the number of consecutive requests has been set to $n = 100$ or $n = 300$. The distribution of requests among locations can be uniform or non-uniform. In each problem instance, the initial server configuration has been specified by hand, while the sequence of requests with a desired length and distribution has been generated automatically by an appropriate random number generator.

The problem instances with $m = 25$ actually correspond to a set of Croatian cities shown in Figure 5, while the instances with $m = 15\,112$ use the map of Germany presented by Figure 6. There is a slight difference in the way how costs of moving among locations are determined in both maps. For Croatian cities, costs have been defined as the shortest distances over the available roads and given to algorithms through a pre-computed distance matrix. On the other hand, German locations have been described by coordinates, so that costs can be computed by algorithms themselves as Euclidean distances. Consequently, there is a slight computational overhead when dealing with the map of Germany.

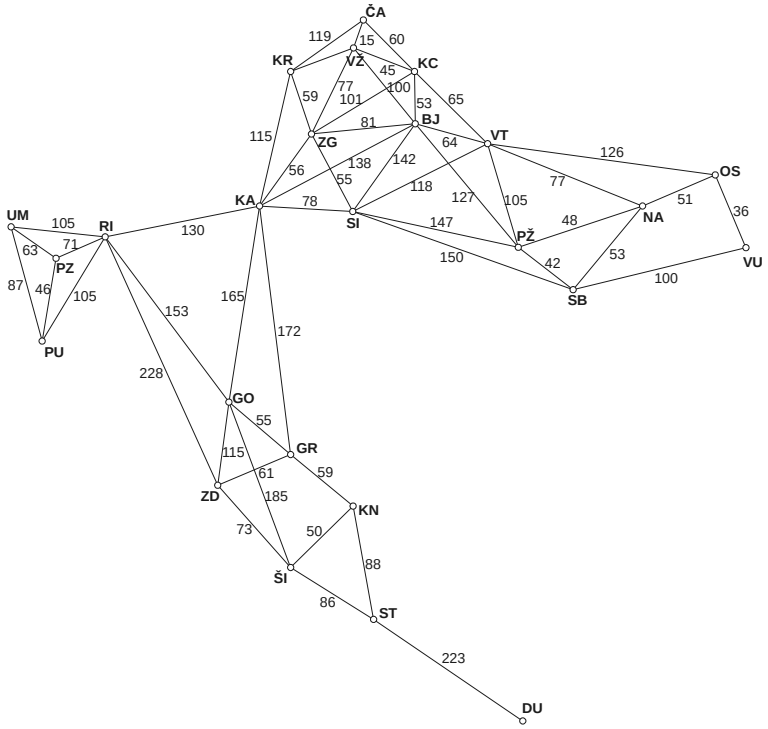


Fig. 5. Map of Croatia

Uniform distribution means that a new request can appear at any location within the considered metric space with the same probability. Non-uniform distribution means that some locations occur more frequently than the others. More precisely, in our non-uniform problem instances each location has a weight and occurs with the probability equal to its weight divided by the sum of all weights. Since locations correspond to cities, their weights can be interpreted as numbers of inhabitants. Thus a new request appears in a city with 50 000 people 5 times more often than in a city with 10 000 people. Non-uniform distribution is more realistic, and it allows algorithms to learn from history.

Experimenting enabled measuring the performance of the w-WFA, as well as its computational complexity. To obtain the desired results, the implemented w-WFA was tested on the described problem instances. More precisely, each instance was solved many times with different window sizes w. In addition, each instance was also solved by GREEDY, BALANCE, the original WFA and OPT. The performance of any algorithm was measured in terms of the incurred total cost of serving, while the computational complexity was estimated as the actual computing time in milliseconds.

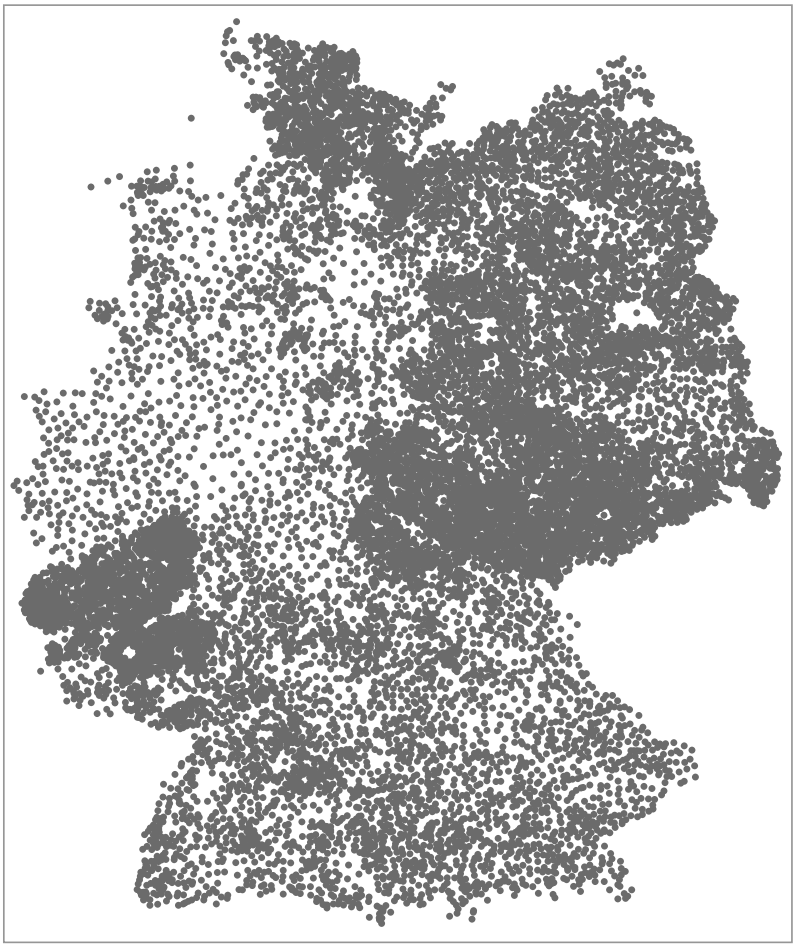


Fig. 6. Map of Germany

The results of our experiments are given in Tables 2-5. Tables 2 and 3 deal with the problem instances based on the map of Croatia ($m = 25$), while Tables 4 and 5 contain the instances that use the map of Germany ($m = 15\,112$). The instances with shorter sequences of requests ($n = 100$) are represented by Tables 2 and 4, while longer sequences ($n = 300$) are put into Tables 3 and 5. In each table, a row corresponds to a particular algorithm, and a column to a particular problem instance. Each table entry records the performance (total cost) and computational complexity (time in milliseconds) of the corresponding algorithm on the corresponding instance.

The most important thing we can see from Tables 2–5 is that the w-WFA usually achieves the same performance as the original WFA if w is large enough. This is

Instance:	#01		#02		#03		#04	
Map:	Croatia		Croatia		Croatia		Croatia	
Locations:	$m = 25$		$m = 25$		$m = 25$		$m = 25$	
Requests:	$n = 100$		$n = 100$		$n = 100$		$n = 100$	
Distribution:	non-uniform		non-uniform		uniform		uniform	
Servers:	$k = 3$		$k = 10$		$k = 3$		$k = 10$	
Algorithm	Total cost	Time [ms]	Total cost	Time [ms]	Total cost	Time [ms]	Total cost	Time [ms]
OPT	2 132	31	707	46	11 431	46	3 215	31
BALANCE	3 279	0	713	0	15 487	0	5 106	0
GREEDY	10 767	0	1 748	0	12 981	0	4 357	15
2-WFA	10 767	15	1 748	79	12 856	16	4 369	235
5-WFA	4 298	16	1 056	78	12 736	47	4 230	359
10-WFA	3 463	16	1 056	125	13 489	62	4 233	610
20-WFA	3 463	47	1 056	250	13 556	204	4 740	1 234
50-WFA	3 463	234	989	843	13 405	1 046	4 772	4 391
Orig WFA	3 463	406	989	1 266	13 405	2 360	4 772	8 828

Table 2. Experimental results – map of Croatia, 100 requests

true even for the biggest problem instances. Thus the w-WFA with a sufficiently large w can be considered equivalent to the WFA in terms of the incurred total cost. After it has been reached for some w, this equivalence is further on retained for larger w-s.

The second important fact visible from Tables 2–5 is that the w-WFA (or equivalently the WFA) frequently achieves better performance than GREEDY or BALANCE. In fact, the w-WFA is always better than GREEDY if the distribution of requests is non-uniform. This phenomenon is easy to explain: non-uniform distribution means that history counts, so that an algorithm that learns from history ought to be better than an algorithm that ignores history. In spite of its simplicity, BALANCE turned out to be more successful than we expected. Still, the w-WFA performs better than BALANCE for uniform distribution of requests.

The data on computing times shown in Tables 2–5 are more or less consistent with the previously presented theoretical estimates of computational complexity. Small anomalies and discrepancies can be attributed to peculiarities of the employed cluster. By comparing Table 2 with Table 4 and Table 3 with Table 5, respectively, we can see that orders of time magnitudes in corresponding columns are roughly equal, which means that computational complexity indeed does not depend on the size m of the metric space M. The times for the large map of Germany still tend to be longer, but this can be explained by the previously mentioned overhead caused by computing Euclidean distances.

By observing each table separately, we can notice that problem instances with non-uniform distribution of requests allow smaller total costs and run faster than corresponding instances with uniform distribution. Indeed, with non-uniform dis-

Instance:	#05		#06		#07		#08	
Map:	Croatia		Croatia		Croatia		Croatia	
Locations:	$m = 25$		$m = 25$		$m = 25$		$m = 25$	
Requests:	$n = 300$		$n = 300$		$n = 300$		$n = 300$	
Distribution:	non-uniform		non-uniform		uniform		uniform	
Servers:	$k = 3$		$k = 10$		$k = 3$		$k = 10$	
Algorithm	Total cost	Time [ms]	Total cost	Time [ms]	Total cost	Time [ms]	Total cost	Time [ms]
OPT	5 847	1 125	895	1 265	35 138	1 140	8 792	1 265
BALANCE	9 440	0	1 531	0	51 546	0	15 405	0
GREEDY	30 836	0	2 985	0	42 789	0	12 164	0
2-WFA	25 706	31	2 985	219	41 969	63	12 433	766
5-WFA	11 857	47	1 227	125	43 032	109	12 562	1 140
10-WFA	10 923	47	1 227	219	41 496	234	11 575	1 860
20-WFA	10 923	156	1 227	453	41 131	625	11 863	3 984
50-WFA	10 891	922	1 229	1 859	41 131	3 875	11 517	15 750
100-WFA	10 891	3 843	1 229	7 500	41 131	16 844	11 805	60 047
150-WFA	10 891	9 641	1 229	30 391	41 131	42 344	11 805	211 266
Orig WFA	10 891	49 375	1 137	74 640	41 131	220 766	11 805	604 718

Table 3. Experimental results – map of Croatia, 300 requests

tribution it happens more often that a new request occurs at a location already covered by a server – then no computing is necessary and the request is served with no cost in time 0.

In accordance with the theoretical estimates, Tables 2–5 clearly demonstrate that the w-WFA runs orders of magnitude faster than the original WFA. This is true even for a fairly large window size w. Note that our tables contain cumulative times needed to serve whole sequences of requests. Big differences in cumulative values are obtained because the w-WFA needs approximately the same time for any step, while the time taken by the original WFA rises with each consecutive step.

More detailed comparison of times needed by our algorithms step by step is given in Figure 7. This figure corresponds to the problem instance #4 (map of Croatia, $m = 25$, $n = 100$, uniform distribution, $k = 10$), and it compares the 50-WFA to the original WFA. The times needed by both algorithms to serve each particular request are presented. As mentioned before, the algorithms recognize situations where a request occurs at a location already covered by a server – then the service time is 0. Otherwise the time fits into one of two functions that are recognizable from the plotted data. For the 50-WFA the corresponding function is bounded by a constant, while for the original WFA the function rises very steeply.

Again in accordance with the theoretical estimates, Tables 2–5 also show that the w-WFA cannot compete in speed with GREEDY or BALANCE. This is true even for a very small window size w. In fact, GREEDY and BALANCE are so quick that our system could not record their times accurately. On the other hand, the time taken by the w-WFA is never negligible.

Instance:	#09		#10		#11		#12	
Map:	Germany		Germany		Germany		Germany	
Locations:	$m = 15\,112$		$m = 15\,112$		$m = 15\,112$		$m = 15\,112$	
Requests:	$n = 100$		$n = 100$		$n = 100$		$n = 100$	
Distribution:	non-uniform		non-uniform		uniform		uniform	
Servers:	$k = 3$		$k = 10$		$k = 3$		$k = 10$	
Algorithm	Total cost	Time [ms]	Total cost	Time [ms]	Total cost	Time [ms]	Total cost	Time [ms]
OPT	53 191	32	30 526	47	414 887	31	221 545	32
BALANCE	91 663	0	34 360	0	572 443	0	351 198	0
GREEDY	99 647	0	76 346	0	437 427	0	270 291	0
2-WFA	99 647	0	76 346	125	437 427	31	269 076	453
5-WFA	99 647	15	36 200	62	446 443	47	263 406	640
10-WFA	76 085	16	36 200	110	451 764	78	268 282	1 047
20-WFA	76 085	16	36 200	203	451 764	235	260 681	2 141
50-WFA	76 085	109	36 200	640	451 764	1 156	260 681	7 344
Orig WFA	76 085	187	36 200	1 047	451 764	2 516	260 681	14 093

Table 4. Experimental results – map of Germany, 100 requests

7 CONCLUSIONS

In this paper we have presented a modified version of the work function algorithm (WFA) for solving the on-line k-server problem. Our modification is based on a moving window with a fixed size. On the other hand, the original WFA can be inter-

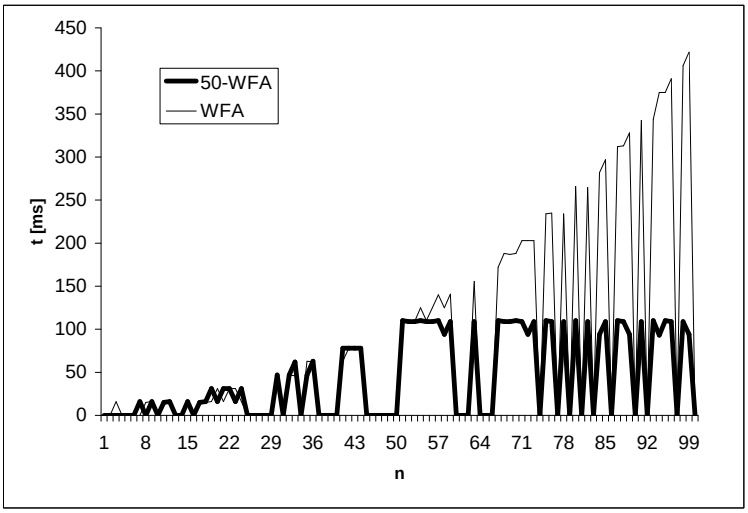


Fig. 7. Detailed results – the original WFA vs. the 50-WFA, time per step

Instance:	#13		#14		#15		#16	
Map:	Germany		Germany		Germany		Germany	
Locations:	$m = 15\,112$		$m = 15\,112$		$m = 15\,112$		$m = 15\,112$	
Requests:	$n = 300$		$n = 300$		$n = 300$		$n = 300$	
Distribution:	non-uniform		non-uniform		uniform		uniform	
Servers:	$k = 3$		$k = 10$		$k = 3$		$k = 10$	
Algorithm	Total cost	Time [ms]	Total cost	Time [ms]	Total cost	Time [ms]	Total cost	Time [ms]
OPT	144 673	1 140	70 052	1 297	1 268 760	1 140	630 284	1 266
BALANCE	203 983	0	100 191	0	1 857 679	0	990 979	0
GREEDY	744 734	0	85 458	0	1 456 535	0	750 735	0
2-WFA	601 824	32	85 458	219	1 443 328	63	750 735	1 344
5-WFA	227 855	31	85 458	312	1 464 790	141	744 422	2 000
10-WFA	227 855	47	78 705	406	1 457 765	281	745 682	3 281
20-WFA	227 855	140	78 705	844	1 466 849	719	754 858	7 047
50-WFA	229 819	860	78 705	3 485	1 466 849	4 609	736 152	28 625
100-WFA	227 539	3 765	78 705	13 078	1 466 849	19 516	734 856	103 953
150-WFA	227 539	10 672	79 091	47 093	1 466 849	50 125	734 856	362 094
Orig WFA	227 539	53 235	79 091	138 219	1 466 849	255 984	734 856	1 071 797

Table 5. Experimental results – map of Germany, 300 requests

preted as an extreme case of our modified WFA where the window size is infinite. We have described how both versions of the WFA can be implemented by using network flow techniques. Also, we have presented a series of experiments dealing with performance and computational complexity of the implemented algorithms.

The presented experimental results have confirmed that the modified WFA can really be regarded as a convenient "lightweight" version of the original WFA. Indeed, with a reasonably large window, the modified WFA closely mimics the performance of the original WFA, i.e. it achieves the same or almost the same incurred total cost of serving. This assertion is true even for very large problem instances. At the same time, the modified WFA runs dramatically faster than the original WFA, thus becoming suitable for practical purposes.

The computational complexity of the modified WFA is still very large compared to simple heuristics, such as the greedy or the balanced algorithm. However, this additional computational effort can be tolerated since it assures better performance. The advantages of the WFA vs. simple heuristics are more visible on problem instances with non-uniform distribution of requests.

In this paper we have also 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 property of competitiveness.

The fact that the modified WFA is not competitive is seemingly in contradiction with the presented experimental results. 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 of non-competitiveness is of course extremely artificial. In fact, the presented experiments show that in realistic situations the modified WFA still captures some advantages of its competitive original, although not being competitive itself.

Our future plan is to develop an optimized and truly distributed network flow implementation of the modified WFA. Since the networks in our network flow problems have quite specific structure, it should be possible to design dedicated algorithms for finding network flows that are more efficient than generic algorithms. Moreover, by employing more processors it should be possible to further speed up the whole computation in order to meet strict response time requirements that are sometimes imposed by on-line problems.

REFERENCES

[1] Bartal, Y.—Grove, E.: The Harmonic k-Server Algorithm is Competitive. Journal of the ACM, Vol. 47, 2000, pp. 1–15.

[2] Bartal, Y.—Koutsoupias, E.: On the Competitive Ratio of the Work Function Algorithm for the k-server Problem. Theoretical Computer Science, Vol. 324. 2004, pp. 337-345.

[3] Baumgartner, A.—Manger, R.—Hocenski, Z.: Work Function Algorithm with a Moving Window for Solving the On-Line k-Server Problem. Journal of Computing and Information Technology, Vol. 15, 2007, pp. 325–330.

[4] Bazaraa, M. S.—Jarvis, J. J.—Sherali, H. D.: Linear Programming and Network Flows. Third edition. Wiley-Interscience, New York, NY, 2004.

[5] Chrobak, M.—Karloff, H.—Payne T. H.—Vishwanathan, S.: New Results on Server Problems. SIAM Journal on Discrete Mathematics, Vol. 4, 1991, pp. 172–181.

[6] Edmonds, J.—Karp, R. M.: Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. Journal of the ACM, Vol. 19, 1972, pp. 248–264.

[7] Irani, S.—Karlin, A. R.: Online Computation. In: D. Hochbaum (Ed.): Approximation Algorithms for NP-Hard Problems. PWS Publishing Company, Boston, MA, 1997, pp. 521–564.

[8] Jungnickel, D.: Graphs, Networks and Algorithms. Springer, Berlin, 2005.

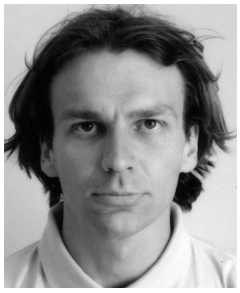
[9] Koutsoupias, E.—Papadimitrou, C.: On the k-Server Conjecture. In: F. T. Leighton and M. Goodrich (Eds.): Proceedings of the 26^{th} Annual ACM Symposium on Theory of Computing, Montreal, Quebec, Canada, May 23–25, 1994. ACM Press, New York, NY, 1994, pp. 507–511.

[10] Manasse, M.—McGeoch, L. A.—Sleator, D.: Competitive Algorithms for Server Problems. Journal of Algorithms, Vol. 11, 1990, pp. 208–230.

[11] Quinn, M. J.: Parallel Programming in C with MPI and OpenMP. McGraw-Hill, New York, NY, 2003.

[12] Sleator, D.—Tarjan, R. E.: Amortized Efficiency of List Update and Paging Rules. Communications of the ACM, Vol. 28, 1985, pp. 202–208.

Alfonzo Baumgartner is a research assistant and a Ph. D. student at the Faculty of Electrical Engineering, Josip Juraj Strossmayer University in Osijek, Croatia. His current research interests include on-line problems, parallel algorithms applied to combinatorial optimization problems, and other algorithms which use efficient data structures. He has published 6 papers in international scientific journals or conference proceedings.



Tomislav Rudec received the M. Sc. degree in mathematics from the University of Zagreb in 2001. Currently, he is a lecturer at the Faculty of Electrical Engineering, Josip Juraj Strossmayer University in Osijek. His research interests include combinatorial optimization, analysis of algorithms, on-line computation, and teaching of mathematics. He has published 4 scientific papers in international journals or conference proceedings, and 6 professional papers.

Robert Manger received the B. Sc. (1979), M. Sc. (1982), and Ph. D. (1990) degrees in mathematics, all from the University of Zagreb. For more than ten years he worked in industry, where he obtained experience in programming, computing, and designing information systems. He is presently a Professor at the Department of Mathematics, University of Zagreb. His current research interests include: combinatorial optimization, parallel and distributed algorithms, and soft computing. He has published 18 papers in international scientific journals, over 25 scientific papers in conference proceedings, 10 professional papers, and 4 course materials. He is a member of the Croatian Mathematical Society, Croatian Society for Operations Research and IEEE Computer Society.