Title

Stefan Caldararu[1] and Marc Renault[2]

[1] *Undergraduate Student with Department of Computer Science, UW-Madison*
[2] *Professor in the Department of Computer Science, UW-Madison*

April 18, 2024

Department of Computer Science
University of Wisconsin – Madison, USA

Abstract

Abstract here.

Keywords: K-Servers , Online Algorithms, Bijective Analysis

Contents

1 Introduction

1.1 Outline

In this paper, we consider the K-Servers problem in a variety of contexts. First, we provide a description of the K-Servers problem, and an in-depth proof of the $2k - 1$ competitiveness of the Work Function Algorithm (WFA). Following this a literature review is provided, focusing on the Unifying Potential for the WFA [1]. We then have an analysis of [1], and provide some thoughts into a potential extension of this work for caterpillar graphs. Following this, we transition to an analysis of other algorithms focusing on the bijective analysis of these algorithms, and describe a C++ environment provided for practical testing of a variety of these algorithms. We provide some experimental analysis of the algorithms within the testing suite, extending the work done in [2].

1.2 Problem Description

An instance of the K-Servers problem can be described by a metric space $M = (X, d)$, a number of servers $k > 1$, and an input sequence $\sigma = (r_1, r_2, r_3, ..., r_n)$, where each r_i corresponds to a point in the metric space. Each of the k servers is assigned an initial starting location within the metric space (generally this is assumed to be the first k requests of the input sequence). Following this, The sequence of reqeusts is processed one at a time. When a request comes in, a given algorithm ALG must decide on one of the servers to service the request. It must then move said server from it's current location x to the request r_i. This incurs a cost of $c = d(x, r_i)$. The goal is to have ALG incur the smallest possible cost while servicing all of the requests in the sequence [3].

There are two major distinctions to be made between different classes of algorithms. The first is the classification of a "lazy" algorithm - one that only moves a server in order to service the current request. Non-lazy algorithms will process a request, and then potentially also move other servers preemptively in order to prepare for future requests. It is important to note that for any non-lazy algorithm that performs well, there is a parallel lazy algorithm that performs just as well, if not better. We can describe this algorithm as follows: suppose we have our non-lazy algorithm, ALG. We have the algorithm $LAZY$ service requests with the same servers that ALG services requests. Suppose that ALG moves a server from location x_1 to location x_2 preemptively, and then later services request r_i with this server. $LAZY$ will be servicing r_i with the same server, except it will be moving directly from x_1 instead of x_2. By the triangle inequality, $d(x_1, r_i) \leq d(x_1, x_2) + d(x_2, r_i)$. By applying this principle throughout the request sequence, we will see that the cost of $LAZY$ will be as good if not better than that of ALG. This shows us that we can create a lazy algorithm from a non-lazy algorithm by maintaining "ghost" locations for servers in relation to how the non-lazy algorithm would use them. Then, we can determine which server the non-lazy algorithm would use, and then service that location with the correct server [3].

The second major distinction is between "online" and "offline" algorithms. An offline algorithm receives the entire request sequence at once, and so as a result is able to make decisions on what server to use for the current request based off of future requests. In contrast, an online algorithm receives the request one at a time, and so is only able to make decisions based off of past requests, the current server configuration, and the current request. While online algorithms are at a severe disadvantage due to this, real world applications often rely on the performance of these algorithms.

Applications range from disk access optimization, such as the two headed-disk problem [3], to police or firetruck servicing.

1.3 Competitive Analysis

The most popular method for analyzing the performance of online algorithms for a problem is competitive analysis. Here, we assume a "malicious adversary", who is attempting to make our algorithm ALG perform as poorly as possible in relation to the performance of the optimal algorithm OPT. The malicious adversary is allowed to come up with any finite input, and our competitive ratio is determined by this. If for any finite input, we are able to guaruntee that ALG has a cost within a certain ratio c of OPT (allowing for a constant additive factor), then we say that our algorithm is c-competitive [3]. So, if we define $ALG(\sigma)$ as the cost ALG incurs while processing request σ, then we have the following definition:

Definition 1.1. Algorithm ALG is said to be ***c*-competitive** if for every finite request sequence σ , $ALG(\sigma) \leq c \cdot OPT(\sigma) + \alpha$ for some constant α.

Competitive analysis is in some sense similar to "worst case" analysis, where we try to see how poorly an algorithm will ever perform. It is worth noting that some algorithms such as a greedy algorithm may perform very well on the majority of inputs, but given specific inputs may not be competitive at all. That is, given some value c, a finite length input can be found such that the algorithm doesn't satisfy the above definition.

Additionally, a lower bound of k has been shown for any online algorithm's competitive ratio. This means that if we are looking at the 3-Servers problem, the best competitive ratio an online algorithm can achieve is 3 [3].

1.4 k competitive lower bound

In this section, we follow a proof for a lower bound of the k server problem, as described in [4].

Lemma 1.1. *For a metric space M where $|M| > k$, no online algorithm ALG can have competitive ratio less than k*

Proof. We begin by restricting all requests to a sub-metric space which has $k + 1$ points. Within this metric space, there are $k+1$ possible configurations of the k servers. We begin by maintaining k offline algorithms (denoted $OPT_1, OPT_2, ..., OPT_k$), where each of these algorithms has a different configuration, and none have the same configuration as ALG. We denote the configuration of ALG as $x_1, x_2, ...x_k \in M$. We begin by requesting the point $r \notin x_1, x_2, ...x_k$. All of the offline algorithms are able to service this request without moving any servers, while ALG must move some server from x_i to service this point. At this point we will have OPT_i, the offline algorithm that has no server at x_i, move its server from r to x_i preemptively. We then repeat this process.

This shows that the cost incurred by ALG will be equal to the sum of the costs incurred by all of the offline algorithms, showing that the competitive ratio of ALG will be at least k. □

2 Algorithms

2.1 Random Algorithm

The random algorithm *RAND* is the most basic of all of the online algorithms, and has little practical use. If the request is not currently covered by a server, then *RAND* randomly selects one of it's servers, and moves the selected server to the service point. This algorithm is mostly just used as a baseline, as we would hope that none of the other algorithms will perform worse than random.

2.2 Greedy Algorithm

The greedy algorithm *GREEDY* is also a computationally inexpensive online algorithm, but which has many practical uses. This algorithm checks the distance that each server would have to travel to get to the service point, and selects the server which would incur the smallest cost. While this algorithm has very good performance for the majority of practical application inputs, it is surprisingly not a competitive algorithm [3]. *GREEDY* is one of the primary focuses of this study, as it is often looked over due to it's non-competitiveness, but still performs very well in practice.

2.3 Optimal Algorithm

Any optimal offline algorithm *OPT* is simply defined as an algorithm that will have the smallest possible cost for any request sequence. Computationally, the fastest implementations leverage a reduction to a Min Cost Max Flow problem. This reduction can be further studied in [5]. This still ends up being a good bit more computationally expensive than the previous algorithms, but is needed to be used as a metric to compare algorithms against, as most algorithms are compared to the optimal when determining their strenghts.

2.4 Work-Function Algorithm

The Work-Function Algorithm *WFA* is considered to be one of the most promising algorithms in terms of the competitive ratio, as it has been proven to be $(2k-1)$-competitive, and is believed to be k-competitive. Given a certain starting server configuration, current configuration, and previous input request sequence, *WFA* determines which server an optimal algorithm would move, while also ensuring that $k-1$ of the servers end up in the current server configuration. This final server is then used to service the current request. A similar reduction to the *OPT* computation can be used to find this server [5]. This means that the *WFA* must compute a Min Cost Max Flow for each request, making it much more expensive than *OPT*, and all of the other algorithms. Additionally, it is also worth noting that this is not a finite memory algorithm, as the WFA must remember all of the previous requests [6].

2.5 Double Coverage Algorithm

The Double Coverage algorithm *DC* is a k-competitive algorithm on the line. If the request is to the left of all of our current server locations, then we just move the left most server to service the request. If the server is to the right of all of the current locations, we use the rightmost server. Otherwise, the request will be between two servers. In this case, we move the two closest servers towards the reqeust at the same rate, until one of the servers reaches the request location. A proof of DC's competitiveness can be found in [3]. It is also important to note that DC is not a lazy algorithm, as it does move more than one server at a time. Additionally, generalizations of this algorithm can be used on metric spaces other than the line, such as trees.

2.6 k-Centers Algorithm

The k-Centers algorithm divides the metric space into k sections, and assigns each server a section of the space to operate in. Implementations of this algorithm can either have the server return to the center of it's operating space after servicing the request in a non-lazy fashion, or just remember the bounds of each operating space, and service each request with the appropriate server [7].

3 Work Function Algorithm

3.1 Algorithm

This section focuses on an in-depth analysis of the WFA. There are two ways of thinking about the WFA's decision-making process. WFA can be thought of as a balancing between retrospective-OPT, and a greedy algorithm. It tries to balance between the decisions OPT would have made up until this point, assuming the current request is the last one that will be made, with a greedy algorithm that attempts to move the least distance possible at the current time step. This helps the algorithm maintain a configuration that is similar to OPT, without moving servers great distances from the current configuration. An algorithmic definition is provided in sec. 3.2.

Additionally, WFA can be thought of from how it's implementation works. WFA makes lazy decisions to move servers, so only moves one server at each step. As described previously, the WFA will attempt to determine the server to move which would have the minimum cost for starting in an initial configuration, and ending with $k-1$ servers in the same locations they are currently in. Then, it will service the request with the server that does not stay in the initial configuration. A description of a min-cut max-flow implementation of this is described in sec. **??**.

3.2 Notations and Definitions

We begin with a couple of notation definitions for clarity.

Definition 3.1. A request sequence σ can be broken up into sub-parts, where σ_i represents the first i requests from within the request sequence

Definition 3.2. A set C of k-points from within the metric space M that the WFA is operating in is called a **configuration**, representing the locations of the servers. Additionally, the initial configuration before any requests have been processed is denoted C_0, and the configuration after the WFA has operated on the first i requests is denoted C_{σ_i}. Additionally, C can be used to denote an arbitrary configuration after servicing some request sequence σ_i, in which case C' denotes the configuration after servicing the next request, i.e. servicing the request sequence σ_{i+1}.

Definition 3.3. The distance between two configurations A and B can be computed as the minimum weight matching between the two sets, denoted $D(A, B)$.

Definition 3.4. The **work function**, denoted $w_\sigma(C)$ computes the minimum value required to begin in configuration C_0, service all requests in σ, and end with servers in configuration C. The work function has similar notation to configurations(w_{σ_i}, w, and w'), except the work function value on an empty request sequence is denoted $w_\emptyset$. Additionally, the work function can be computed as follows:

$$w_\emptyset(C) = D(C_0, C)$$
$$w'(C) = min_{x \in C}\{w(C - x + r) + d(x, r)\}$$

Definition 3.5. The **work function algorithm** moves the server currently at point s at each step, incurring a cost $d(s, r)$. This server is determined as follows:

$$s = argmin_{x \in C}\{w(C - x + r) + d(x, r)\}$$

3.3 $2k - 1$ Proof

In this section, we follow the proof showing that the WFA is $2k - 1$ competitive from [3], and add some in-depth explanatory details. It is worth noting a few basic property of the work function:

$$w'(C') = w(C') = w'(C) - d(s, r) \tag{1}$$

$$w'(C) = min_{x \in C}[w'(C - x + r) + d(r, x)] = min_{x \in C}[w(C - x + r) + d(r, x)] \tag{2}$$

The first part of property 1 follows because $r \in C'$. The second part assumes that s is the point from which the server that moved to service r, and follows from the definition of the WFA.

The first part of 2 says that the work done to service a request sequence *including* the next request r for some configuration C is equal to the minimum work done to service the same request sequence, except on the configuration C without the point x and with the point r, plus the distance between r and x. The set $C - x + r$ is defined as in [3], where there is only one copy of r in this set, whether or not r was originally in C.

The nice part about this is that we now *know* that $r \in C - x + r$, which isn't necessarily true of C. This means that the second part of this equation is true, where we can remove the r from the work function, as it is already included in the set.

Definition 3.6. The **offline pseudocost** $w'(C') - w(C)$ is the work done to service a request sequence σ_{i+1} and end in configuration C', minus the work done to service a request sequence σ_i and end in configuration C.

The offline pseudocost is a measure of how much work is done to service the next request in the sequence, given that the current configuration is C. If we sum this across all requests on sets C_0, C_1 ... C_n for the configuration of OPT, we will see that this will telescope to $w_\sigma(C_n) - w_\emptyset(C_0)$, where $\sigma_0 = \emptyset$. This is equal to the cost $OPT(\sigma)$, as we are subtracting the cost to get into the initial configuration C_0 from the final work function $w_\sigma(C_n)$.

Definition 3.7. The **extended cost** is defined as $max_X\{w'(X) - w(X)\}$.

The extended cost represents the worst case configuration we could be in given the next request. That is, the configuration that will maximize the cost incurred to satisfy the next request. The extended cost represents the maximum cost we could incurr by servicing the next request online.

Definition 3.8. A work function is **Quasi-Convex** (QC) if for all configurations X, Y, and for all $x \in X$, the following property holds:

$$min_{y\in Y}\{w(X - x + y) + w(Y - y + x)\} \leq w(X) + w(Y)$$

Quasi-Convexity essentially says that for every two configurations, the sum of the costs to get into those configurations is greater than or equal to the *best* cost of swapping *some* point $y \in Y$ with x. The key point here is that there will be some minimizing configurations X and Y. We can additionally take this a step further with the following definition.

Definition 3.9. A work function is **General Quasi-Convex** (GQC) if for every X, Y, there exists a bijection $X \to Y$ such that for all partitions of X into X_1 and X_2, the following property holds:

$$w(X_1 + g(X_2)) + w(g(X_1) + X_2) \leq w(X_1) + w(X_2)$$

It is worth noting that if a work function is GQC, then it is also QC. This is because we can set $X_1 = X - x$ and $y = g(x)$. This will give us the QC property.

Lemma 3.1. *If a bijection g satisfies the GQC property, then there exists a bijection $\tilde{g}$ that satisfies the GQC property such that $\tilde{g}(x) = x$ for all $x \in X \cap Y$.*

Proof. Suppose we have a bijection $g : X \to Y$ that satisfies the GQC property, and additionally of all such bijections, maps the *maximum* number of elements in $X \cap Y$ to themselves. by contradiction, assume that there exists some $a \in X \cap Y$ such that $g(a) \neq a$. Now, we define $\tilde{g} : X \to Y$ such that:

$$\tilde{g}(x) = \begin{cases} g(x) & \text{if } x \neq a \text{ or } x \neq g^{-1}(a) \\ a & \text{if } x = a \\ g^{-1}(a) & \text{if } x = g^{-1}(a) \end{cases}$$

Now, let (X_1, X_2) be a partition of X, and without loss of generality, $g^{-1}(a) \in X_1$. If $a \in X_1$, then $g(X_1) = \tilde(g)(X_1)$, and $g(X_2) = \tilde{g}(X_2)$, and so $\tilde(g)$ satisfies the GQC property, which is a contradiction to our second assumption. Therefore, $a \notin X_1$, and so we have:

$$w(X) + w(Y) \geq w((X_1 + a) \cup g(X_2 - a)) + w(g(x_1 + a) \cup (X_2 - a))$$

By the definition of GQC. Then, since $a, g^{-1}(a) \notin X_2$, we know that $g(X_2 - a) = \tilde(g)(X_2 - a)$, by the definition of $\tilde{g}$. Additionally, $g(X_1 + a) = \tilde{g}(X_1 + a)$, since $a, g^{-1}(a) \in X_1$. Therefore, we have:

$$\begin{aligned} w((X_1 + a) \cup g(X_2 - a)) + w(g(x_1 + a) \cup (X_2 - a)) &= w((X_1 + a) \cup \tilde{g}(X_2 - a)) + w(\tilde{g}(X_1 + a) \cup (X_2 - a)) \\ &= w(X_1 \cup \tilde{g}(X_2)) + w(\tilde{g}(X_1) \cup X_2) \end{aligned}$$

We achieve the last equality because $a \in X_1$ and $a \notin X_2$, so $X_1 - a = X_1$ and $X_2 - a = X_2$. This would mean that $\tilde{g}$ satisfies the GQC property, which is again a contradiction. Therefore, there exists a bijection $\tilde{g}$ that satisfies the GQC property such that $\tilde{g}(x) = x$ for all $x \in X \cap Y$. □

Lemma 3.2. *All work functions are GQC, and so are therefore also QC.*

Proof. We prove this by induction on the length of the request sequence. Our base case has $i = 0$, where $w_{\emptyset}(X) + w_{\emptyset}(Y) = D(C_0, X) + D(C_0, Y)$. We consider two minimum weight matchings, whose values are $D(C_0, X)$, and $D(C_0, Y)$. Each $c_j \in C_O$ is mapped by M_X to some $x_j \in X$ and by M_Y to some $y_j \in Y$. We show that the bijection $g(x_j) = y_j$ satisfies the GQC property. Because X and Y are minimum weight matchings, $w(X) + w(Y)$ is an upper bound of $D(X, Y)$. By the difinition of g, we maintain the indeces of our points when transitioning $X \to C_0 \to Y$. By a pointwise approach, we see that we satsify the GQC property with equality. We can think of this as doubling the number of servers at locations in C_0 and then matching them to the points in X and Y. Each point in X and Y will appear exactly once on either side of the equation.

For the induction step, we assume w satisfies the GQC property, and we now have a new request r. We show that w' satisfies GQC. We know that there exists some $x \in X$ such that $w'(X) = w(X-x+r)+d(x,r)$, and similarly there exists some $y \in Y$ such that $w'(Y) = w(Y-y+r)+d(y,r)$. We know there exists a bijection $g : (X - x + r) \to (Y - y + r)$ that satsfies the GQC property, where $g(r) = r$. We define $\tilde{g} : X \to Y$ as follows:

$$\tilde{g}(z) = \begin{cases} g(z) & \text{if } z \neq r \\ y & \text{if } z = r \end{cases}$$

From here, we will have $X_{xr} = X - x + r$ and $Y_{yr} = Y - y + r$ for easier notation. Without loss of generality, assume that $x \in X_1$ for some partition X_1, X_2 of X. We have:

$$\begin{aligned} w'(X) + w'(Y) &= w(X_{xr}) + w(Y_{yr}) + d(r,x) + d(r,y) \\ &= w((X_1 - x + r) \cup g(X_2)) + w(Y_{yr}) + d(r,x) + d(r,y) \\ &\geq w((X_1 - x + r) \cup g(X_2)) + w(g(X_1 - x + r) \cup (X_2)) + dx, r + dr, y \\ &= w(X_{xr} \cup \tilde{g}(X_2)) + w((\tilde{g}(X_1) - y + r) \cup X_2) + d(x,r) + d(r,y) \\ &\geq w'(X_1 \cup g'(X_2)) + w'(g'(X_1) \cup X_2) \end{aligned}$$

□

Definition 3.10. Let w be the current work function and let r be any point in our metric space. A configuration A is called a **minimizer of** r **with respect to** w if:

$$A = argmin_{(X)}\{w(X) - \Sigma_{x \in X} d(x,r)\}$$

Lemma 3.3. *Let w be the current work function and let r be the next request. If A is a minimizer of r with respect to w, then A is also a minimizer of r with respect to w'.*

Proof. We want to show that for every B, $w'(A) - \Sigma_{a \in A} d(a,r) \leq w'(B) - \Sigma_{b \in B} d(b,r)$. This is equivalent to showing the following, as per property 2:

$$min_{a' \in A}\{w(A - a' + r) + d(a', r) - \Sigma_{a \in A} d(a,r)\} \leq min_{b' \in B}\{w(B - b' + r) + d(b', r) - \Sigma_{b \in B} d(b,r)\}$$

Since A is a minimizer of r with respect to w:

$$w(A) - \Sigma_{a \in A} d(a,r) \leq w(B - b' + a') - \Sigma_{b \in B - b' + a'} d(b,r)$$

$$w(A) - w(B - b' + a') - \Sigma_{a \in A} d(a,r) \leq -\Sigma_{b \in B} d(b,r) + d(b',r) - d(a',r)$$

$$-w(B - b' + a') + w(A) + d(a',r) - \Sigma_{a \in A} d(a,r) \leq -\Sigma_{b \in B} d(b,r) + d(b',r)$$

Now, we apply the GQC Lemma, with $X = B - b' + r$, $Y = A$, $x = r$, and $y = a'$. This means that $min_{a' \in A}\{w(B - b' + a') + w(A - a' + r)\} \leq w(B - b' + r) + w(A)$. We now sum the two sides of this with the previous inequality to get the following equation:

$$\begin{gathered} min_{a' \in A}\{w(B - b' + a') + w(A - a' + r) - w(B - b' + a') + w(A) + d(a',r) + \Sigma_{a \in A} d(a,r)\} \\ \leq w(B - b' + r) + w(A) + d(b',r) - \Sigma_{b \in B} d(b,r) \end{gathered}$$

And after cancelations, we get that for all $b' \in B$:

$$min_{a' \in A}\{w(A - a' + r) + d(a',r) - \Sigma_{a \in A} d(a,r)\} \leq w(B - b' + r) + d(b',r) - \Sigma_{b \in B} d(b,r)$$

Since this is true for all $b' \in B$, our property holds for the b' that minimizes the right hand side, completing our proof. □

Definition 3.11. We define the **extended cost** as the cost of the value achieved by the configuration which is able to maximize the difference between the next work function value and the current work function value:

$$max_X\{w'(X) - w(X)\}$$

Definition 3.12. Additionally, we define a **maximizer with respect to** w as the configuration A which achieves this extended cost value:

$$A = argmax_X\{w'(X) - w(X)\}$$

Lemma 3.4. *Any minimizer with respect to r is also a maximizer with respect to w.*

Proof. Suppose A is a minimizer with respect to r. We want to show that for every B,

$$w'(B) - w(B) \leq w'(A) - w(A)$$

$$w'(B) + w(A) \leq w'(A) + w(B)$$

If $r \in B$, then this statement is true, as $w'(B) = w(B)$, and $w(A) \leq w'(A)$ by definition.

By expanding the above desired equation using eq. 2, we get that we want to show:

$$min_{b' \in B}\{w(B - b' + r) + d(b',r) + w(A)\} \leq min_{a' \in A}\{w(A - a' + r) + d(a',r) + w(B)\}$$

This is equivalent to saying that for all B, $a' \in A$:

$$min_{b'\in B}\{w(B-b'+r)+d(b',r)+w(A)\} \le w(A-a'+r)+d(a',r)+w(B)$$

To show this, we begin with the statement that A is a minimizer. In particular, this means that:

$$w(A)-\Sigma_{a\in A}d(a,r) \le w(A-a'+b')-\Sigma_{a\in A-a'+b'}d(a,r)$$

$$w(A)-\Sigma_{a\in A}d(a,r) \le w(A-a'+b')-\Sigma_{a\in A}d(a,r)+d(a',r)-d(b',r)$$

$$w(A)+d(b',r)-d(a',r) \le w(A-a'+b')$$

When we apply the QC property with $X = A-a'+r$, $Y = B$, $x = r$, and $y = b'$, and then substitute in for $w(A-a'+b')$, we get:

$$min_{b'\in B}\{w(A-a'+b')+w(B-b'+r)\} \le w(A-a'+r)+w(B)$$

$$min_{b'\in B}\{w(A)+d(b',r)-d(a',r)+w(B-b'+r)\} \le w(A-a'+r)+w(B)$$

$$min_{b'\in B}\{w(B-b'+r)+d(b',r)+w(A)\} \le w(A-a'+r)+d(a',r)+w(B)$$

□

Definition 3.13. We now define the **value of a minimizer configuration**, for minimizer A of r with respect to w as:

$$MIN_w(r) = w(A)-\Sigma_{a\in A}d(a,r)$$

We now fix any optimal algorithm OPT, and have $U = \{u_1, u_2, ..., u_k\}$ as the current configuration of OPT. We will now have the potential function: $\Phi(U,w) = \Sigma_{u\in U}MIN_w(u)$. Additionally, suppose that for the next request r, OPT services the request with server u_j. Then, the next configuration of OPT is $U' = U-u_j+r$.

Lemma 3.5. $\Phi(U',w)-\Phi(U,w) \ge k*d(u_j,r)$

Proof. Using the triangle inequality, we know that for all a, $d(a,r) \le d(a,u_j)+d(u_j,r)$. Putting the value of a minimizer together with this, we get:

$$MIN_w(r) = min_A\{w(A)-\Sigma_{a\in A}d(a,r)\}$$
$$\ge min_A\{w(A)-\Sigma_{a\in A}[d(a,u_j)+d(u_j,r)]\} = MIN_w(u_j)-k*d(u_j,r)$$

$$\Phi(U',w)-\Phi(U,w) = \Sigma_{u\in U'}MIN_w(u)-\Sigma_{u\in U}MIN_w(u)$$
$$= MIN_w(r)-MIN_w(u_j) \ge MIN_w(u_j)-k*d(u_j,r)-MIN_w(u_j) = -k*d(u_j,r)$$

□

Lemma 3.6. $\Phi(U',w')-\Phi(U',w) \ge max_X\{w'(X)-w(X)\}$

Proof. Let A be a minimizer of r with respect to w. Again, we know that $w'(X) \geq w(X)$ for all X, as any sequence of moves that would service σr and end in configuration X would also service requests σ and be able to end in X with the same or lesser distance traveled. Therefore, for any point p, we have:

$$MIN_{w'}(p) = min_X\{w'(X) - \Sigma_{x\in X} d(p, X)\} \geq min_X\{w(X) - \Sigma_{x\in X} d(p, X)\} = MIN_w(p)$$

This means that MIN is also monotone incresing with respect to the request sequence. We have:

$$\begin{gathered} \Phi(U', w') - \Phi(U', w) = \Sigma_{u\in U'}(MIN_{w'}(u) - MIN_w(u)) \\ = MIN_{w'}(r) - MIN_w(r) + \Sigma_{u\in U'-r}(MIN_{w'}(u) - MIN_w(u)) \\ \geq MIN_{w'}(r) - MIN_w(r) \end{gathered}$$

Because A is a minimizer of r with respect to w, we know by Lemma 3.3 A is also a minimizer of r with respect to w'. So by using the duality lemma 3.4, we have:

$$\begin{gathered} \Phi(U', w') - \Phi(U', w) \geq MIN_{w'}(r) - MIN_w(r) \\ = w'(A) - \Sigma_{a\in A} d(a, r) - [w(A) - \Sigma_{a\in A} d(a, r)] = w'(A) - w(A) = max_X\{w'(X) - w(X)\} \end{gathered}$$

□

Lemma 3.7. $\Phi(U_\sigma, w_\sigma) \leq K * OPT(\sigma)$

Proof. By the definition of MIN, we have:

$$MIN_{w_\sigma}(u) =\leq min_{y\notin U_\sigma}\{w_\sigma(U - u + y) - \Sigma_{x\in U-u+y} d(x, u)\}$$

As the configuration U with $u \in U$ minimizes the terms inside of the minimization on the right hand side. Suppose y^* is a valid y that minimizes the right hand side of the above equation. We have:

$$\begin{gathered} MIN_{w_\sigma}(u) \leq w_\sigma(U - u + y^*) - \Sigma_{x\in U-u+y^*} d(x, u) \\ \leq w_\sigma(U - u + y^*) - d(y^*, u) \leq w_\sigma(U_\sigma) = OPT(\sigma) \end{gathered}$$

When we sum over all $u \in U_\sigma$, we complete the lemma by the definition of Φ

$$\Phi(U_\sigma, w_\sigma) = \Sigma_{i=1}^k MIN_{w_\sigma}(u_i) \leq K * OPT(\sigma)$$

□

Lemma 3.8. *For some constant α independent of k,*

$$\Phi(U_\emptyset, w_\emptyset) \geq -\Sigma_{a,b\in C_0} d(a, b) = \alpha$$

Proof. The sum in the middle of the above equation is clearly a constant that only depends on the initial configuration, and so therefore depends on the metric space, not directly on the number of servers. By definitions of Φ and MIN, we have:

$$\Phi(U_\emptyset, w_\emptyset) = \Sigma_{u \in U_\emptyset} MIN_{w_\emptyset}(u) = \Sigma_{u \in U_\emptyset} min_X\{w_\emptyset(X) - \Sigma_{x \in X} d(x, u)\}$$

Suppose X^* is the configuration of servers which minimizes the right hand side for some $MIN_{w_\emptyset}(u) = min_X\{w_\emptyset(X) - \Sigma_{x \in X} d(x, u)\}$. Without loss of generality, assume that u_i matches to x_i in the minimum weight matching between the two sets. Then, $w_\emptyset(X^*) = D(U_\emptyset, X^*)$. By using the triangle inequality, we get:

$$MIN_{w_\emptyset}(u) = \Sigma_{i=1}^k d(u_i, x_i) - \Sigma_{i=1}^k d(x_i, u)$$
$$= \Sigma_{i=1}^k [d(u_i, x_i) - d(x_i, u)] \geq \Sigma_{i=1}^k [d(u_i, x_i) - (d(u_i, x_i) + d(u_i, u))] = -\Sigma_{i=1}^k d(u_i, u)$$

$$\Sigma_{u \in U_\emptyset} MIN_{w_\emptyset(u)} \geq -\Sigma_{u \in U_\emptyset} \Sigma_{i=1}^k d(u_i, u)$$
$$= -\Sigma_{a,b \in C_0} d(a, b)$$

□

Definition 3.14. We define the **total extended cost** as the sum over all moves of the extended costs.

$$TEC = \Sigma_{i=0}^{n-1} max_X\{w_{i+1}(X) - w_i(X)\}$$

Lemma 3.9. *If the following property holds for some constant α independent of σ, then WFA is c-competitive:*

$$TEC \leq (c+1) * OPT(\sigma) + \alpha$$

The above lemma follows directly by the provided definitions, because the total extended cost bouds the cost incurred by the work function algorithm. This lemma means that we are now only required to show the following:

Lemma 3.10. *$TEC \leq 2k * OPT(\sigma) + \alpha$, and therefore the WFA is $2k-1$ competitive.*

Proof. By putting together lemmas 3.5 and 3.6, we get:

$$\Phi(U', w') - \Phi(U, w) \geq max_X\{w'(X) - w(X)\} - k * d(u_j, r)$$

When we sum across all requests in the request sequence, we get the telescoping sum:

$$\Phi(U_\sigma, w_\sigma) - \Phi(U_\emptyset, w_\emptyset) \geq TEC - k * OPT(\sigma)$$

By using lemmas 3.7 and 3.8, we get:

$$k * OPT(\sigma) + \Sigma_{a,b \in C_0} d(a, b) \geq \Phi(U_\sigma, w_\sigma) + \Sigma_{a,b \in C_0} d(a, b)$$
$$\geq \Phi(U_\sigma, w_\sigma) - \Phi(U_\emptyset, w_\emptyset) \geq TEC - k * OPT(\sigma)$$

$$TEC \leq 2k * OPT(\sigma) + \Sigma_{a,b \in C_0} d(a, b) = 2k * OPT(\sigma) + \alpha$$

□

3.4 Min-Cut Max-Flow Computation

In this section, we describe a non-recursive implementation of the work function algorithm utilizing a min cost flow problem formulation, as described in [8]. A similar approach can be used for an implementation of an optimal offline algorithm, and is described in the cited artice.

Min-Cut Max-Flow Formulation

For a given work function $w_{r_1,r_2...r_i}(C_0, C_i)$ we have four groupings of nodes, each representing different structures from the underlying work function algorithm, along with a source node s and a sink node t. The first set of nodes are labeled $S = \{s_1...s_k\}$ representing the initial server configuration, C_0. We additionally have a set of nodes $S' = \{s'_1...s'_k\}$ representing the final configuraiton, C_i. The last two sets, $R = \{r_1...r_i\}$ and $R' = \{r'_1...r'_i\}$ together represent the request sequence. Each node in S is connected to the source node s with a weight of 0. Now, each node in S is connected to each node in S' with a weight of $d(s_j, s'_k)$ the distance between the two points in the metric space. Each node in S is also connected to each node in R with a weight of $d(s_j, r_k)$. Each node $r_j \in R$ is connected *only* to it's corresponding node $r'_j \in R$, witha sufficiently large negative weight (i.e. $-L$). Each node $r'_j \in R' - r'_i$ is connected to each $r_k \in R$ where $k > j$, with a weight of $d(r_j, r_k)$. Additionally, each node $r' \in R' - r'_i$ is connected to each node $s'_k \in S'$ with a weight of $d(r'_j, s'_k)$. The node r'_i is connected to t with a weight of 0. Finally, we compute a value X such that:

$$X = \frac{1}{k-1}\Sigma_{j=1}^{k} d(s'_j, r_i)$$

And connect each $s' \in S'$ to t witha weight of $X - d(s', r_i)$. Each arc in this graph has a maximum flow of 1, clearly making the min-cut equal to k (if we cut between s and nodes in S). This flow can be seen in an example network in fig. 1.

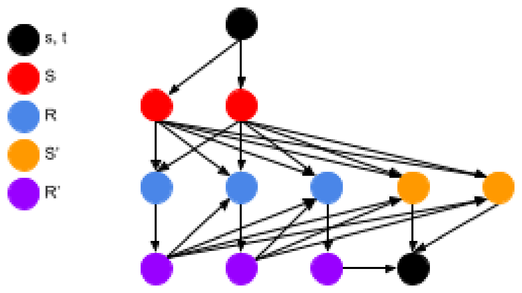

Figure 1: Min-Cut Max-Flow for Work Function with k=2 and i=3

Explanation of the formulation

When the maximum flow is achieved, we are guarunteed to have flow passing through every node in S. The min cost ensures that we ahve flow passing through every node in R and R', because of the large negative weight between the two sets. Finally, since r'_i is only connected to t, there will be some node $s' \in S'$ that has no flow going to t. This will indicate the optimal server to move to service the request r_i. Since each flow in the network is 1, we can break down the network and follow each flow individually. A flow passing through some server $s_j \in S$ indicates the server j that starts at this initial location. If this server passes through some request r_k to r'_k, this indicates the server j servicing the kth request. Finallly, the server will either go to some node s' in the current configuration, or will go to r'_i, indicating that it is the server which services the next request. It is worth noting that this provides a solution to *one* step of the work function. Whenever we get a new request, we must reconstruct this graph with the additional request added, and recompute a new solution.

Finding the cost

We follow the process described in [8] for solving the minimum cost flow problem. We begin with a null flow through the graph, and then proceed with k iterations each adding a flow of 1 traversing through the graph.

Each iteration, we find the minimum cost flow through the graph using a modified Bellman-Ford algorithm to ensure that we both get the minimum cost possible, and the capacity of the flow is not exceeded for any arc. We then reverse both the flow and the cost along the arcs traversed by this flow for all following iterations. This process is repeated until we have k flows through the graph. This means that when the first flow is passed through, all nodes in R and R' will be passed through and have their flows and costs reversed, as this minimizes the cost (due to the large negative weight between R and R'). Following this, flows may go directly from nodes in S to nodes in S', or they may go through nodes in R and R', but will not re-pass through the r_j to r'_j connections, as these will have very large costs associated with them.

4 Unifying Potential

4.1 Unifying Potential Function Description

In this section, we describe a unifying potential [1] that is a promising candidate for proving that the WFA is k competitive. This potential is used to show that various subcases (ex. $k = 2$, $k = n-1$, $k = n-2$, and various specialized metric spaces) are k competitive. While many of these subcases have been shown to be k competitive in the past [4, 9–12], the novelty of this approach is a *single* potential function that is able to prove all of these cases. The authors of this unifying potential work take this as indication that it may be possible to prove the general k competitiveness for the WFA using this potential.

Definition 4.1. The **diameter** Δ of a metric space M is defined as the largest distance between any two points in the metric space.

$$\Delta = max_{x,y \in M}\{d(x,y)\}$$

Definition 4.2. A point $\tilde{x} \in M$ is called an **antipode** of another point $x \in M$ if for every $y \in M$, $d(x, y) + d(y, \tilde{x}) = \Delta$.

It is worth noting that every metric space can be easily extended such that every point has an antipode. This is done by defining a copy of the original metric space, $\tilde{M}$, where for every point $x \in M$, we have a copy $\tilde{x} \in \tilde{M}$. We denote the diameter of the original metric space Δ. Distances within $\tilde{M}$ and between the two spaces are defined as follows:

$$d(\tilde{x}, \tilde{y}) = d(x, y)$$

$$d(\tilde{x}, y) = 2\Delta - d(x, y)$$

It is worth noting that the diameter of this extended metric space will be 2Δ, and every point will have an antipode. Additionallay, on an extended metric space $M \cup \tilde{M}$ where every point $p \in M$ has an antipode $\tilde{p} \in \tilde{M}$, for any point $q \in M$ and any point $\tilde{r} \in \tilde{M}$, $d(p, q) \leq d(p, \tilde{r})$. This just states that the cost for traversing to a node on the current subgraph you are on is *always* less than or equal to the cost of traversing to the other subgraph.

Definition 4.3. For a metric space M where every point $x \in M$ has an antipode $\tilde{x}$, we define the function $\Phi_{x_1,x_2,...,x_k}$ for a given configuration $x_1, x_2, ..., x_k$ as:

$$\Phi_{x_1,x_2,..x_k}(w) := \Sigma_{i=0}^{k} w(\tilde{x_i}^i, x_{i+1}, ..., x_k)$$

where x^i represents i copies of the point x.

Definition 4.4. We define the **potential** $\Phi(w)$ as:

$$\Phi(w) := min_{x_1,x_2,...,x_k \in M} \Phi_{x_1,x_2...,x_k}(w)$$

Lemma 4.1. *For a metric space M where every point has an antipode, If for every request $r \in M$ and for every work function w_0, w_1, ..., w_n it holds that $\Phi(w) = \Phi_{x_1,x_2,...,x_k}(w)$ for some $x_1, x_2, ...x_k \in M$ where $x_k = r$, then the WFA is k competitive on the metric space M.*

While a proof for this is not provided here, we reference [1], where a complete proof of this lemma is provided. This involves definition of a $n - k$ evader potential $\tilde{\Phi}$, which is the dual to this problem. In this formulation, you have $n - k$ evaders in a metric space that must move away from requested points. It is clear that a solution to the evader problem would yield a k server solution.

4.2 Useful Insights

In this section, we provide a couple of useful insights that have been noted while looking at the unifying potential. These range from general properties of work functions, to specific properties for the unifying potential. Additionally, we go over a few of the proofs provided in [1].

General Work Properties

Lemma 4.2. *Work functions are 1-Lipschitz, i.e. for any work function and any two configurations X, Y on any metric space, the following property holds:*

$$w(X) - w(Y) \leq d(X, Y)$$

Proof. This follows directly from the triangle inequality, and is more self explanatory in the following form:

$$w(X) \leq w(Y) + d(X, Y)$$

This says that the *worst* value for the work done to get into configuration X would be the work done to get into configuraiton Y, plus the cost to transfer between the two configurations. □

Definition 4.5. It is said that configuration Y **supports** X if the following property holds:

$$w(X) - w(Y) = d(X, Y)$$

Lemma 4.3. *A lazy optimal algorithm OPT has no reason to be in a configuration that is supported by another configuration.*

Proof. Suppose OPT is in a configuraiton X that is supported by another configuration Y after servicing some request r_i. Suppose points in X and Y are indexed according to their minimum cost matching, i.e. x_j matches with y_j in the matching. We can define a new offline algorithm, OPT_1, which ends in configuration Y after servicing r_i. Following this, Whenever OPT services a request r with the jth server located at x_j, OPT_1 will service the request with the same server, located at y_j. We know that $d(y_j, r) \leq d(y_j, x_j) + d(x_j, r)$, and OPT has already incurred the cost $d(y_j, x_j)$ (by the definition of support). Therefore, the cost incurred by OPT_1 will be less than or equal to that incurred by OPT. □

This provides some useful insights, and a new way of looking at this problem. Since at each step, we know that OPT can only be in a limited set of configurations (the set of configurations that are not supported by other configurations), we must only "keep up" with those.

Definition 4.6. A **tree graph** is a set of nodes that are connected by edges which is acyclic (No path of unique edges leads back to the starting node), and satisfies the metric property. Additionally, a **leaf node** is one that is only connected by *one* edge to the rest of the graph.

Lemma 4.4. *For any metric space M that can be represented as a tree graph, there exists a minimizer A of r with respect to w such that every $x \in A$, x is a leaf node.*

Proof. Suppose A is a minimizer of r with respect to w, with $x \in A$ such that x is not a leaf node. We will show that for some leaf node l, $A - x + l$ is a minimizer of r with respect to w.

Because x is not a leaf node, it must be connected to at least two other nodes. This means that there exists some node y that is connected to x by an edge, where $d(x, r) + d(x, y) = d(y, r)$. Therefore, we have the following:

$$\begin{aligned} w(A - x + y) - \Sigma_{a \in A - x + y} d(a, r) &= w(A - x + y) - \Sigma_{a \in A} d(a, r) + d(x, r) - d(y, r) \\ &= w(A - x + y) - \Sigma_{a \in A} d(a, r) + d(x, r) - d(x, r) - d(x, y) \\ &= w(A - x + y) - \Sigma_{a \in A} d(a, r) - d(x, y) \leq w(A) - \Sigma_{a \in A} d(a, r) \end{aligned}$$

Where the final inequality holds by eq. 4.2. □

4.3 Unifying Potential on the Caterpillar Graph

5 Bijective Analysis

5.1 Additional Algorithms

5.2 Bijective Analysis Description

In this section, we present a few additional methods for analysis on algorithms other than competitive analysis. These methods are useful for comparing algorithms to each other, and can be used to demonstrate an algorithms efficiency outside of the "worst-case" considered by competitive analysis. Competitive analysis considers all requests of finite length. For each of these methods described below, we take an experimental-based approach. That is, each analysis method will be considerind the performance of an algorithm on a given request sequence length, and so can therefore be directly calculated in practice.

Direct Analysis

Direct analysis is a similar techinque to competitive analysis, in that we directly compare the performance of ALG to the performance of OPT on a request. Rather than looking across all request sequences of any finite length, we determine a specified length for our request sequence. Then, we look directly at the ratio of ALG's to OPT's performance for each input, and take the largest such ratio. It is important to note that as the length of our input approaches infinity, our direct analysis ratio will aproach the competitive ratio of our algorithm.

Definition 5.1. Algorithm ALG has a direct analysis ratio of c if for every input σ of length n, $ALG(\sigma) \leq c \cdot OPT(\sigma)$.

Max/Max Ratio

For the Max/Max ratio, we are comparing the worst case of each algorithm to each other. In practice, this makes sense to do on finite sets of input sequences. Here, we will take the highest cost of ALG and the highest cost of OPT, and this ratio will be our performance metric [6].

Bijective Analysis

The bijective ratio is similar to using direct analysis, except we allow for a bijection between the two data sets. So, we look at the input space of all inputs of length n, denoted I_n. If there exists a bijetion $\pi : I_n \to I_n$ such that $ALG(\sigma) \leq c \cdot OPT(\pi(\sigma))$, then we say that the bijective ratio between ALG and OPT is c [7]. Therefore, we obtain def. 5.2.

Definition 5.2. For a given input space I_n, we say that algorithm A has bijective ratio c with respect to algorithm B if there exists a bijection $\pi : I_n \to I_n$ such that $A(\sigma) \leq c \cdot B(\pi(\sigma))$, $\forall \sigma \in I_n$.

It is important to note that this definition doesn't only compare the ratio of an algorithm to the optimal, but can also be used to compare between two different online algorithms. This can be used to prove an online algorithms optimality [7], and allows for some interesting test-case results found in sec. **??**.

5.3 C++ Implementations

In this section, we describe the software package developed as part of this work. The package is written in C++, and has various levels of performance capabilities. It can additionally be run using the Lemon Graph Library [13], or with purely internal data structure implementations. The various implementaiotns allow users to run the various algorithms described in sec. **??** either with a single thread, or under a producer-consumer framework for parallel processing. Additionally, a finite memory implementation is provided, and a memoization approach on the input sequence is implemented allowing for increased efficiency. Finally, example code for running the software on a High Throughput Cluster using Condor [14] is provided.

Basic structures

The most basic structure in the software package is the `Mspace` class. This class uses a `std::vector` to store the graph as an adjacency matrix of distances between nodes. Functions such as `setSize`, `getSize`, `getDistance`, and `setDistance` allow for easy manipulation of the metric space.

`getInput` and `writeOutput` classes are written to help maintain RAII standards, and allow for easy reading and writing of input and output files.

An abstract `Alg` class is provided as a framework for each of the various algorithms to build off of. This class defines all basic functionality for an algorithm to run, and has a virtual function `run` that must be implemented by each algorithm, which takes an input request sequence and returns the cost incurred by the algorithm. It contains a `Mspace` object, and stores the current configuration of the servers in two forms: a `config` vector which stores the integer location of each server, as well as a `coverage` vector that stores whether or not each location in the metric space is covered by a server. This allows for efficient checking of whether or not a location is covered by a server, as well as efficient access to where the servers are located. `setServers` and `setGraph` functions are provided to allow for easy initialization of the algorithm, and a `moveServer` function is provided for easy manipulation of the server configuration.

5.4 Experimental Results

6 Conclusion

References

[1] C. Coester and E. Koutsoupias, “Towards the k-server conjecture: A unifying potential, pushing the frontier to the circle,” 2021.

[2] S. Caldararu and M. Renault, “An analysis of k-server algorithm performance,” 2023.

[3] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis.* Cambridge University Press, 1998.

[4] E. Koutsoupias, “The k-server problem,” *Computer Science Review*, vol. 3, pp. 105–118, 2009.

[5] T. Rudec, A. Baumgartner, and R. Manger, “A fast work function algorithm for solving the k-server problem,” *Central European Journal of Operations Research*, vol. 21, pp. 1–19, 01 2009.

[6] S. Ben-David and A. Borodin, "A new measure for the study of on-line algorithms," *Algorithmica*, vol. 11, pp. 73–91, 2005.

[7] S. Angelopoulos, M. P. Renault, and P. Schweitzer, "Stochastic dominance and the bijective ratio of online algorithms," 2016.

[8] T. Rudec, A. Baumgartner, and R. Manger, "A fast work function algorithm for solving the k-server problem," *Central European Journal of Operations Research*, vol. 21, pp. 1–19, 2011.

[9] M. Chrobak and L. Larmore, "The server problem and on-line games," vol. 7, 01 1991.

[10] E. Koutsoupias and C. Papadimitriou, "The 2-evader problem," *Information Processing Letters*, vol. 57, pp. 249–252, 1996.

[11] B. Yair and E. Koutsoupias, "On the competitive ratio of the work function algorithm for the k-server problem," *Theoretical Computer Science*, vol. 324, pp. 337–345, 2004.

[12] W. Mein, M. Chrobak, and L. Larmore, "The 3-server problem in the plane," *Theoretical Computer Science*, vol. 289, pp. 335–354, 2002.

[13] E. Software, "LEMON – library for efficient modeling and optimization in networks." `https://lemon.cs.elte.hu/trac/lemon`, 2024.

[14] HTCondor Team, "HTCondor: High-throughput computing software." `https://htcondor.org/`, 2024.