The CNN Problem and Other k-Server Variants

Elias Koutsoupias[1]

Department of Informatics, University of Athens, Athens, Greece and Computer Science Department, University of California, Los Angeles, USA

David Scot Taylor

Department of Computer Science, San José State University, San José, California, USA

Abstract

We study several interesting variants of the k-server problem. In the CNN problem, one server services requests in the Euclidean plane. The difference from the k-server problem is that the server does not have to move to a request, but it has only to move to a point that lies in the same horizontal or vertical line with the request. This, for example, models the problem faced by a crew of a Certain News Network trying to shoot scenes on the streets of Manhattan from a distance; for any event at an intersection, the crew has only to be on a matching street *or* avenue. The CNN problem contains as special cases two important problems: the BRIDGE problem, also known as the cow-path problem, and the weighted 2-server problem in which the 2 servers may have different speeds. We show that any deterministic online algorithm has competitive ratio at least $6 + \sqrt{17}$. We also show that some successful algorithms for the k-server problem fail to be competitive. In particular, no memoryless randomized algorithm can be competitive.

We also consider another variant of the k-server problem, in which servers can move simultaneously, and we wish to minimize the *time* spent waiting for service. This is equivalent to the regular k-server problem under the L_∞ norm for movement costs. We give a $\frac{1}{2}k(k+1)$ upper bound for the competitive ratio on trees.

Email addresses: elias@di.uoa.gr (Elias Koutsoupias), taylor@cs.sjsu.edu (David Scot Taylor).

[1] Supported in part by NSF grant CCR-0105752.

1 Introduction

Consider a CNN crew trying to shoot scenes on Manhattan. As long as they are on a matching street or avenue, they can zoom in on a scene. If a scene happens to be at an intersection, the crew has two choices: street or avenue. Of course, the crew must make its choice online, without knowing where the subsequent scenes will be.

This is an example of an interesting variant of the k-server problem. We can formulate the CNN problem as follows: there is one server in the plane which services a sequence of requests (points of the plane). To service a request $r = (r_1, r_2)$, the server must align itself with the request either horizontally or vertically, i.e, it must move to a point of the vertical line $x = r_1$ or a point of the horizontal line $y = r_2$. The goal is to minimize the total distance traveled by the server. In the online version of the problem the requests are revealed progressively.

A more general formulation of the CNN problem results by assuming that we have 2 servers, each moving in a metric space independent of the other server. Given two metric spaces M_1 and M_2, with one server in each, a request is a pair of points (x_1, x_2) with $x_i \in M_i$. To service the request, we have to move only one server to the requested point of its space. We will call this problem the *sum* of two 1-server problems. The CNN problem is the special case where both metric spaces M_1 and M_2 are lines.

Let $\tau_1, \tau_2, \ldots, \tau_n$ be task systems [7] (not necessarily distinct). These task systems can be combined to get two new interesting online (task system) problems: the *sum* and the *product* of $\tau_1, \tau_2, \ldots$. Given a request for each task system, to satisfy the sum of the requests, at least one of them must be serviced. To satisfy the product of requests, all of them must be serviced. When all task systems are identical, $\tau_i \equiv \tau$, the product is related to randomized online algorithms for the task system τ. A deterministic algorithm for the product of n copies of τ, with each request the same across all spaces, is equivalent to a randomized algorithm against an oblivious adversary with exactly n (equiprobable) random choices; these algorithms are called barely random, or mixed strategies, in the literature [6].

The CNN problem belongs to the class of sum problems. It is a very simple sum problem, which may act as a stepping stone towards building a robust (and less ad hoc) theory of online computation. After quite a few years[2] of intense interest by the research community, a recent breakthrough was achieved in [21] where it was shown that there exists an online algorithm with a finite, albeit

[2] The CNN problem was originally proposed by Mike Saks and William Burley, who obtained some initial results. The name (CNN) was suggested by Gerhard Woeginger.

very high, competitive ratio. The CNN problem and more generally the sum of online problems give flexibility to model problems which the k-server problem cannot, without forcing one to the full task system model. For instance, while the k-server problem has been used to model the behavior of multiple heads on a disk, the CNN problem can be used to model retrieving information which resides on multiple disks. This, for example, happens when we replicate data to achieve higher performance or fault tolerance [1,5,20]. Each disk may have information in completely different locations, leading to independent costs for information retrieval. The goal is to minimize time spent looking for data; which disk the information comes from is not important. In contrast, writing must be performed to all disks; this is closer in spirit to the product online problem mentioned above.

We use *competitive analysis* [4,15,22] to evaluate the quality of online algorithms; the competitive ratio is defined to be the worst-case performance of the online algorithm, compared to the optimal cost for the same sequence of requests. More precisely, algorithm ALG is c-competitive if there is a constant α such that over any finite input sequence ρ, $\text{ALG}(\rho) \leq c \cdot \text{OPT}(\rho) + \alpha$, where $\text{OPT}(\rho)$ is the optimal cost for ρ. The game-theoretic structure of the competitive ratio suggests considering the online algorithm as a strategy that competes against an optimal "adversary", who selects the requests and services them too.

In this work, we show some negative results (lower bounds and failed attempts to "import" the k-server theory to this problem). There are several orders of magnitude between our lower bound and the recently proven competitive ratio of 10^5 [21]. Compare this with the k-server problem: although the k-server conjecture has not been resolved yet, we now know the competitive ratio within a factor of 2 [16]. In particular, the 2-server problem was settled from the beginning [18]. The CNN problem seems very similar to the 2-server problem, yet almost all known competitive algorithms for the k-server problem fail for the CNN problem.

We start by proving a simple tight lower bound for the competitive ratio for the sum of any k non-trivial spaces (spaces with at least 2 distinct points each). Next, we observe that if we restrict the requests to a line, the problem is equivalent to the *weighted* 2-server problem [13] in a line. The weighted k-server problem is the variant of the standard k-server problem in which each server has a weight, and the cost to move a server is its weight times the distance moved. We show (Theorem 2) that any deterministic online algorithm has competitive ratio at least $6 + \sqrt{17}$ for the weighted 2-server problem in a line (and thus for its generalization, the CNN problem). This lower bound holds when one server is arbitrarily faster than the other. We also show that some obvious candidate algorithms which are competitive (or even optimal) for the 2-server problem are not competitive for the CNN problem. For instance, there is no competitive memoryless randomized algorithm for the CNN problem, as discussed in Section 4. Some of the results extend directly to the CNN problem in higher dimensions (the sum of k 1-server problems), in which the lower bound of $k^{\Omega}(k)$ from [13] also applies.

Another way of formulating the weighted server problem is to consider the servers to have different speeds, where cost is proportional to the time needed to service a request. This motivates another interesting variant of the k-server problem, in which we try to minimize the *time* spent waiting for service, instead of trying to minimize the cost of moving servers. Here, we allow multiple servers to move simultaneously, as in [12]. When a request is made, the online algorithm specifies possible movement for each of the k-servers, and tries to minimize the cost of their total movement, under the L_∞ norm. (The cost to move each server is determined by its metric space, but to combine server movements, the L_∞ norm is used.) For k-servers in a tree, we determine the exact ratio $\frac{1}{2}k(k+1)$ of the DC-TREE algorithm of [8,10]. In particular for $k = 2$, we show that DC-TREE is optimal with competitive ratio 3.

This paper expands upon a preliminary version in [17].

2 Lower Bounds for k Spaces

We begin with a simple lower bound for the sum of 1-server problems on k spaces.

Theorem 1 *In the sum of any k 1-server problems, in which each metric space has at least 2 points, no online algorithm has a competitive ratio less than $2^k - 1$.*[3] *There exist metric spaces for which this bound is tight.*

PROOF. To show the lower bound proof we simply extend the proof in [18] which shows that the k server problem has competitive ratio at least k. Given k spaces with 2 points in each (with arbitrary, non-zero distances between the points), we restrict attention to 2 fixed points on each metric space —all requests will be on these points and we can assume that the online servers will also remain on these points. There are 2^k possible configurations for the k servers on those points, of which the online algorithm will be in one

[3] This lower bound applies differently than the $k^{\Omega}(k)$ lower bound from [13], which discusses the server problem when the servers have different speeds. In their terminology, their proof requires different "speeds", while our bound holds even if all servers have the same speed. On the other hand, the sum problem allows for queries not possible in their model. Server speeds will be further discussed in Section 3.

configuration. This online algorithm will play against $2^k - 1$ adversaries, each in one of the other possible configurations.

The online configuration defines exactly one request (one point on each metric space) that forces the online algorithm to move and consists exactly of the points not occupied by online servers. When the online algorithm moves a server for the request, exactly one of the adversaries —the one in the new online configuration— matches its movement in reverse. The total cost of all $2^k - 1$ adversaries is equal to the cost of the online algorithm which shows that the algorithm has competitive ratio at least $2^k - 1$.

To show that this bound is tight for some spaces, consider the simplest non-trivial metrics: each of the k spaces has 2 points, one unit apart. In this metric, there exists a 2^k-1-competitive online algorithm. Again, at any point in time, we can assume that the adversary requests the unique request which does not intersect the online configuration; otherwise the online algorithm incurs no cost. The online algorithm can use any strategy which will progress through each of the 2^k possible configurations in turn, with only one server moving at each step —any Gray code will do. Any offline algorithm must move once within any block of $2^k - 1$ movements by the online servers. □

3 Server Problems with Different Speeds

We now turn our attention to the restricted CNN problem where all requests are from a line (the server is still allowed to move anywhere in the plane). A lower bound for the restricted problem is naturally a lower bound for the unrestricted one. Without loss of generality, we assume that the line is of the form $y = mx$, for some constant m. For $m = 1$, the problem is equivalent to the standard 2-server problem in a line, where moving in the x dimension corresponds to moving one of the servers, and the y dimension the other. Changing m gives a more interesting problem: if all requests are restricted to the form (x, mx), it corresponds to a request in a line at x for a 2-server problem, but this time the servers have different costs for movement. Loosely, this can be interpreted as having servers with different speeds[4] and we wish

[4] To make this interpretation strict, we consider that each request is revealed at the instant when the previous one is serviced, and we allow only one server to be moved at a time. This latter restriction is natural for our problem motivation (we can only move along streets or avenues). It is also worth considering this question without this restriction, which corresponds to using the L_∞ norm instead of L_1 to calculate the cost of combined server movements. The k-server variant of this problem, where servers are allowed to move simultaneously, was introduced in [12] as the min-time server problem. We study this in Section 5.

to minimize the total delay, i.e., the time requests wait for service.

In general, the restriction of the multidimensional CNN problem where all requests are from a line is equivalent to the *weighted k-server problem* in a line. The general weighted server problem was studied in [13]. This work gives a lower bound of $k^{\Omega(k)}$ for any metric space with at least $k+1$ points (and arbitrary weights or speeds). No upper bound is known for arbitrary metric spaces, but [13] gives a doubly exponential upper bound ($2^{2^{O(k)}}$) for uniform metric spaces; this is reduced to exponential ($k^{O(k)}$) when the servers have at most 2 different speeds.

For the restricted version of the CNN problem, the weighted 2-server problem in a line, we show a deterministic lower bound of $6+\sqrt{17}$. The surprising fact exploited in the proof is that the adversary can force the slower server to "simulate" the BRIDGE (or cow-path) problem [2,19]. Thus, the CNN problem contains as a subproblem another fundamental online problem. Interestingly, we know of two different ways to view the BRIDGE problem as a special case of the CNN problem.

The BRIDGE problem is a simple online problem, in which an explorer comes to a river. There is a bridge across the river, but it is not known how far away it is, or if it is upstream or downstream. The explorer must try to find the bridge while minimizing movement. The optimal solution involves alternating between the upstream and downstream directions, exploring 1 distance unit downstream, then 2 units (from the original starting position) upstream, then 4 downstream, and continuing in powers of 2 until the bridge is found. This strategy results in total movement no more than 9 times the distance from the original position to the bridge (plus a constant if the bridge starts out very close to the origin in the opposite direction from the first guess). This competitive ratio is optimal.

Theorem 2 *For the weighted 2-server problem in a line, when moving one server costs m times as much as the other, for sufficiently large m, the deterministic competitive ratio is at least $6+\sqrt{17}$.*

PROOF. We will show that as m grows large, the bound can be forced arbitrarily close to $6+\sqrt{17}$ in stages. We first show a weaker lower bound of 9 to exhibit the relation between the CNN problem and the BRIDGE problem. The role of the explorer will be played by the slow server.

Without loss of generality, we can assume that the online algorithm is lazy (moves a server only to service requests). Let $[l,r]$ be the interval of the line explored (visited so far) by the slow server. Initially, it is safe to assume that the slow server is at $r=0$ and the fast server is at $l=-1$. When the online slow server is at point r the adversary's strategy distinguishes two cases: if

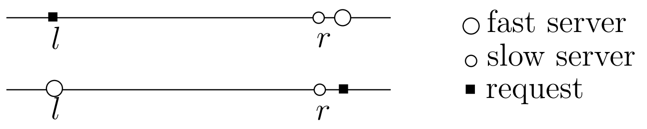

Fig. 1. Bridge problem simulation phase: while the slow server is at its rightmost point, the adversary chooses its next request depending on the location of the fast server. For these requests, a lazy online algorithm will always be in one of these two positions, or one of two symmetric cases.

the fast server is to the right of r, the next request is at l; if the fast server is to the left of r, the next request is at $r + \delta$, where δ is an arbitrarily small positive distance (Figure 1). The adversary's strategy when the slow server is at l is symmetric.

This adversary's strategy forces the slow server to explore a larger and larger portion of the line. The fast server cannot endlessly service requests, as this would lead to an unbounded ratio. Thus, some of the requests must be answered by the slow server. However, the slow server at r cannot move exclusively to the right, as this would result in a competitive ratio of m when compared to the offline strategy of simply moving the fast server to the right, while the slow server sits at position l. To achieve ratio less than m, the slow server must eventually service a request at point l. The adversary can continue to force the slow server to "zig-zag", exploring larger and larger segments of the line; thus the slow server mimics the explorer in the BRIDGE problem.

At the end of the game, the define the interval explored by the slow server to be $[-z, y]$. Consider the that the slow server has just moved from $-z$ to y. Since the competitive ratio of the BRIDGE problem is 9, the total distance moved by the slow server must be at least $9y$ (minus an insignificant term). On the other hand, the adversary can service all requests by moving its slow server to y and its fast server to $-z$. Its total cost is $y + z/m$, which for large m is approximately y. (Strictly speaking, m should be fixed before the online algorithm is forced to choose z values. Here, we can assume that $z < 4y$, otherwise a ratio of greater than 11 can be achieved in the BRIDGE problem.)

Thus, the competitive ratio is at least $9 - \epsilon$, where ϵ tends to 0 as m tends to ∞.

By accounting for the cost of the fast online server, we can improve the bound to $10 - \epsilon$. Let x_0 be the cost of the fast online server. Besides the strategy above of moving its fast and slow servers in opposite directions to service all requests, the adversary has an alternative strategy: service all requests with the fast server. Let $r \geq 9$ be the ratio from just the slow server in the BRIDGE portion of the simulation above. The total cost of the offline algorithm is no more than $x_0 + ry/m$: for these requests, it can mimic the movement of both online servers with this cost or less. Thus, the competitive ratio is at least

$$\max_{x_0,y}\left(\frac{ry+x_0}{y+z/m},\frac{ry+x_0}{x_0+ry/m}\right).$$

(The first term comes from the preceding proof of the 9 lower bound. The second term comes from the alternate adversary strategy.) As above, $z<4y$, or else a ratio greater than 11 can be achieved in just the BRIDGE phase. As m grows, z/m and ry/m become insignificant when compared to y, for all $r \le 6+\sqrt{17}$. (If $r > 6+\sqrt{17}$, the adversary can halt after the BRIDGE phase.) With this given, we can eliminate the z/m and ry/m terms from the denominators, at which point it becomes clear that to minimize the ratio above, the smallest possible value for r is best, that is, $r=9$. (The optimal online algorithm may not follow the optimal bridge strategy above, but for the sequence of requests given here, the best possible online strategy will be one which optimizes the BRIDGE phase. Such an algorithm must have a competitive ratio of $6+\sqrt{17}$ on the requests here.)

The simplified equation is

$$\max_{x_0,y}\left(\frac{9y+x_0}{y},\frac{9y+x_0}{x_0}\right).$$

To minimize the above term, x_0 should be chosen to make the the two values equal, which happens at $x_0=y$, giving a ratio of 10. In the original equation, the ratio approaches 10 from below as the value of m increases, that is, it gives $10-\epsilon$ for $\epsilon=\Theta(1/m)$. Henceforth, to simplify equations, we will eliminate terms which vanish as $m\to\infty$, and thus eliminate the ϵ terms.

Let ρ be the sequence of requests described above (which completes the "BRIDGE simulation"). After the ρ requests, the online server has its slow server at y, and its fast server at $-z$, and these positions match the positions of one of the adversary strategies above. However, the alternative adversary strategy finishes with its slow server at 0 and the fast server at y. If requests now alternate between 0 and $-z$, all such requests can be serviced by the offline algorithm by moving its fast server from y to $-z$. After a sufficiently large number of requests at 0 and $-z$ (say j_1 such pairs of requests) the online slow server must move to 0, or else the online algorithm will incur unbounded costs. (Moving it to $-z$ passes through 0, so the equations below still hold.) Let x_1 be the cost paid by the online fast server shuttling between 0 and $-z$ before this occurs. With this possibility for the adversary, the ratio is:

$$\max_{x_0,x_1,y}\left(\frac{9y+x_0}{y},\frac{10y+x_0+x_1}{x_0}\right).$$

Clearly, x_1 cannot be less than 0. Setting the terms equal for the remaining two parameters, the only positive root occurs at $x_0 = y(-4 + \sqrt{26})$, which results in a ratio of $5 + \sqrt{26}$.

To prove a lower bound of $6 + \sqrt{17}$, we extend this idea of adding extra requests to the end of the BRIDGE sequence. Consider the request sequence $\rho((0, -z)^{j_i}(y, -z)^{k_i})^n$ where i varies from 1 to n, and j_i (k_i) refers to the number of times the first (second) pair are repeated during the i^{th} repetition of the whole phrase. Let j_i and k_i, be determined as follows: after servicing ρ,

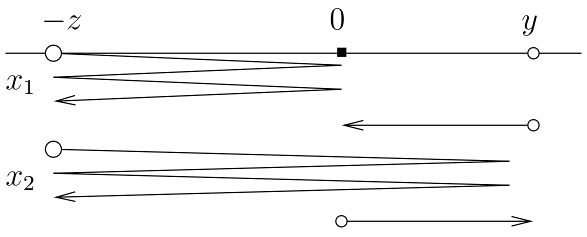

Fig. 2. After the Bridge Simulation: following the initial BRIDGE phase, a series of requests at 0 and $-z$ force the slow server to return to the origin. After it returns to the origin, another sequence of requests at $-z$ and y eventually force it to return back out to y. The total cost of the fast server in each phase is x_i.

the online slow server is at y. To service $(0, -z)^{j_1}$, it may use the fast server for a while, but eventually, it must move its slow server to 0. (If the slow server ever moves to z, it will incur a worse ratio than the lower bound proven here. We can assume that $z > \frac{4}{3}y$, otherwise a ratio of greater than 11 can be achieved in just the BRIDGE phase. Should the online server ever choose to move from 0 to $-z$ (or y to $-z$), the adversary can immediately stop the sequence. In this case, one of the two adversary strategies[5] already already seen will achieve a higher ratio than the one proven here.)

Let j_i be the number of repetitions of $(0, -z)$ needed to make the online slow server move to 0 for the i^{th} time, and k_i be the number of repetitions of $(y, -z)$ needed to make the online slow server move back to y for the i^{th} time. Let $x_{2i-1} = j_i z/m$, the movement of the online fast server before the slow server's

[5] In fact, for large n, the offline servers have a third main strategy to consider: moving the slow server to point z and servicing the rest of the queries with the fast server. Given this possibility, the online server must in fact occasionally move its slow server to the point $-z$. Thus, we do not expect that the bound we prove here is tight. Preliminary calculations show that allowing this third option with the same adversary request sequence will not increase the lower bound by more than a small constant, no larger than 0.22. Solving the resulting equations precisely is complicated by the fact that with this option, it is no longer safe to assume that the optimal BRIDGE solution for ρ will give the best solution to the global equations. We prove our bound for the case when the adversary, at the completion of the BRIDGE phase, guarantees the online algorithm that its slow server is not at, and never will be at, $-z$. Thus the online algorithm need not minimize its ratio against this possible offline strategy.

i^{th} move to 0, and $x_{2i} = k_i(z+y)/m$, the movement of the online fast server before the slow server's i^{th} move back to position y. (For the $5+\sqrt{26}$ bound above, we have only 1/2 of one phase, that is, we have a j_1 term but no k_1 term.)

The adversary might end the game after any of the slow server moves. The first few terms of the ratio R are:

$$\max_{x_i,y}(\frac{9y+x_0}{y}, \frac{10y+x_0+x_1}{x_0}, \frac{11y+x_0+x_1+x_2}{y+x_1}, \frac{12y+\sum_{i=0}^{3} x_i}{x_0+x_2}, \ldots),$$

and in general:

$$R \geq \max_{x_i,y}(\frac{(9+2j)y+\sum_{i=0}^{2j} x_i}{y+\sum_{i=1}^{j} x_{2i-1}}, \frac{(10+2j)y+\sum_{i=0}^{2j+1} x_i}{\sum_{i=0}^{j} x_{2i}}).$$

The different denominator types are from the two offline strategies we have already seen for the ρ (BRIDGE phase) requests: moving both servers, or just the fast one.

The expression above is minimized when all values are equal, and the equations simplify to:

$$\frac{x_0+x_1+10y}{x_0} = \frac{x_0+9y}{y},$$
$$\forall i \geq 2,\ \frac{x_{i-1}+x_i+2y}{x_{i-1}} = \frac{x_0+9y}{y}.$$

Scaling y to be 1, and solving, we get:

$$x_1 = x_0^2 + 8x_0 - 10$$

$$\forall i \geq 2,\ x_i = (x_0+8)^{i-1}x_1 - 2\sum_{j=0}^{i-2}(x_0+8)^j.$$

Simplifying the geometric series and substituting in x_1, this final equation gives

$$\forall i \geq 2,\ x_i = (x_0+8)^{i-1}(x_0^2+8x_0-10-\frac{2}{x_0+7}) + \frac{2}{x_0+7}.$$

All x_i values must be positive, so the smallest possible value for the equations is when $x_0^2+8x_0-10-\frac{2}{x_0+7} = 0$. The only positive root is $x_0 = \sqrt{17}-3$, which gives the stated bound of $6+\sqrt{17}$. $\square$

Leaving our 2-servers in a line interpretation behind, this $6+\sqrt{17}$ lower bound for the CNN problem can also be achieved by considering requests which lie in the two lines $y=0$ and $y=1$. For this special case of the CNN problem, a slightly weaker lower bound was obtained by William Burley and the first author (unpublished). Here, any request can be satisfied at a cost of 1 by moving from one line to the other. We use a strategy similar to the previous one. Suppose the leftmost and rightmost positions of our server so far have been $[l, r]$. If the server is on $y = 0$, place a request at $(l - 1, 1)$, and if the server is on $y = 1$, place a request at $(0, r + 1)$. These requests again make the server move away from the origin simulating the BRIDGE explorer as before. The vertical movement here corresponds to the movement of the fast server in the previous argument. As before, after the simulation of the bridge problem, at the end the adversary can force the server to alternate between the origin and the shorter "arm" of exploration. All equations are the same. (There does not seem to be a natural third strategy for the adversary as in Footnote 5.)

To see just how difficult it is to find a competitive ratio for the CNN problem, we notice that some simple algorithms which are competitive for the 2-server problem are not competitive for the case when the servers have different movement costs. The "Double Coverage" (DC) algorithm in a line is the following simple algorithm: if the request is between the two servers, move both towards it until the request is served. Otherwise, move the closer server (ties broken arbitrarily.) The "Balance", or BAL algorithm is also simple: to answer any query, move the server which will have the minimum cumulative cost if it moves to the request. More general balance algorithms base their decision to move a server to a request on two parameters: the cumulative cost of a server and the distance to the request.

For two servers in a line, one with speed 1 and the other $m \geq 1$, neither algorithm has a constant competitive ratio bounded independently of m. We expect that this statement holds even allowing for obvious modifications needed to the algorithms to account for the different speeds of the servers: for instance, in DC it must be possible for the fast server to "pass" the slow server for requests outside of their convex hull, or else it is trivial to achieve competitive ratio at least equal to m.

We give a a simple example as an intuitive justification for the failure of DC. For a fixed m, consider an online configuration $(0, x)$, where the fast server is at 0, and x is large enough that the slow server can reach a request at $x - 1$ before the fast one ($(x - 1) > m$ will do for the most natural generalization of DC.) By repeating the sequence of requests $0, x, 0, x - 1$, the online servers will pay cost 2 for every 4 requests, while an adversary could satisfy them with just cost $2/m$, by having its server positions reversed. By making m large, we can get an arbitrarily large competitive ratio. This is in stark contrast to DC for the regular k-server problem, in which it is k-competitive, for any k, and can be extended to work optimally even in trees. A similar example can be used for BAL. In this case, the fast server will occasionally be used to answer a request at x or $x + 1$, but will then return to answer the request at 0.

4 Memoryless Randomized Algorithms

In contrast to the natural memoryless algorithm HARMONIC for the k-server problem [3,14]), we proved in [17] that no competitive memoryless online algorithms exist for the CNN problem. Independently, [11] proved that no such algorithms exist for the weighted 2-server problem in a line, which is a subproblem of the CNN problem as shown in Section 3, by limiting requests to the line $y = mx$. Rather than repeat the proof from [17], we refer the reader to [11], due to the brevity and simplicity of their proof.

We note that while the proof from [17] uses a stronger definition of "memoryless" than that from [11], it does prove something something slightly stronger than the fact that there is no finite competitive ratio: it proves that with a single fixed cost move, an adversary can force an expected unbounded cost for any memoryless online algorithm.

5 The k-Server Problem under the L_∞ Norm

In this section, we consider the k-server problem, where servers have the same speed, but can be moved simultaneously, and the objective is to minimize the *time* of service. Once a request is served, the next request is revealed. It is simple to achieve a competitive ratio which is k times larger than the ratio of the regular k-server problem —the online algorithm can simply move one server at a time, forfeiting its choice to move multiple servers simultaneously. Using the best known bound of [16], this gives a $2k^2 - k$ ratio upper bound, but we expect that this can be improved. In the uniform metric space, simply moving the servers in order will achieve the (optimal) ratio of k.

We show that in a tree, the competitive ratio is no worse than $\frac{1}{2}k(k+1)$. We employ the DC-TREE algorithm of [10], generalized from DC of [8]. The algorithm is defined as follows: move each of the servers with an unblocked (by other servers) path to the request towards the request at a constant speed. Note that servers may begin moving, and later stop moving as they become blocked by other servers which move onto their path.

Theorem 3 *For k servers in a tree under the L_∞ norm,* DC-TREE *has competitive ratio $\frac{1}{2}k(k+1)$.*

PROOF. The analysis is as in [10], with only a slight change in the potential function. Let M_{min} be the distance for the best matching between the online and offline servers, and ΣDC-TREE be the distance between all unordered pairs of online servers. The argument uses the following potential function:

$$\Phi = \frac{(k+1)M_{min}}{2} + \frac{1}{2}\Sigma\text{DC-TREE}.$$

With an offline move of cost d, offline server can increase M_{min} by kd, thus increasing the potential by $\frac{k(k+1)d}{2}$.

Next, consider a time in which x online servers move distance d: at most $x-1$ of them move away from their matched offline server, while one moves towards its match, so M_{min} increases by at most $d(x-2)$. The moving servers all move towards each other, a distance of $2d$ each pair. At most one server moves away from each stationary online server, and $x-1$ servers move towards each stationary online server, a distance of of d each. So, ΣDC-TREE decreases by $2dx(x-1)/2 + d(k-x)(x-2)$, and the total change in Φ is at least $d(x-2)(k+1)/2 - dx(x-1)/2 - d(k-x)(x-2)/2 = -d$, while d is the cost to the online algorithm.

To show this ratio is tight for DC-TREE, consider servers (online and offline) at position $(2, 4, \ldots, 2k)$ of a line. For a cost of 1, the offline algorithm can move all of its servers to $(1, 3, \ldots, 2k-1)$. The adversary is lazy, and will at each time request its uncovered server which is at the lowest value (i.e., the sequence request will be $1, 3, 1, 5, 3, 1, 7, 5, 3, 1, \ldots$). Each request will cost DC-TREE 1 to serve, and it will take $\frac{1}{2}k(k+1)$ total requests to converge to the offline position, at which time we are at a position similar to the original one. $\square$

Unfortunately, under the L_∞ norm, the DC-TREE algorithm for $k = 2$ can not be extended to arbitrary metric spaces as it is in [9] under the L_1 norm. There, "virtual" movements can be remembered and performed later, without additional costs. Here, free (non-maximum) movements (real or virtual) must be made immediately or else they are no longer free.

For the case of $k = 2$, DC-TREE is optimal, as the following lemma shows.

Lemma 4 *For the 2-server problem, where the L_∞ norm is used for each movement cost, no online algorithm has competitive ratio less than 3 for two servers in a line.*

PROOF. Consider requests in a line, with initial configuration (online and offline) $(0, 2)$. If there is a request at 1, by symmetry, the online algorithm can service it by moving the server at 2, and it can move the other server to any point in the interval $[-1, 1]$ at no extra cost. Next the adversary requests point 3, and can reveal its configuration $(1, 3)$. The online algorithm must pay at least 2 to service the second request, and can be forced by repeated requests to 1 and 3 to move to configuration $(1,3)$. The total online cost is at least 3, but the offline cost is 1 (move the 2 servers together from $(0,2)$ to $(1,3)$). This configuration is similar to the initial one, and the situation can be repeated indefinitely. Therefore, no online algorithm has ratio less than 3. □

6 Conclusions and Future Work

We have introduced several interesting variants of the k-server problem, with the power to model new problems. There are numerous open problems left. We mention only a few of them here.

In [21], a finite competitive ratio of 10^5 has been proven for the general 2-server problem (the sum of any two 1-server problems), which includes the CNN problem, confirming our original conjecture that it has a finite competitive ratio. Clearly the gap can be narrowed. We believe that the actual ratio for CNN is a small constant —less than 20. In fact, we conjecture that the generalized Work Function Algorithm (which moves the server which minimizes $\lambda w(A') + d(A, A')$) has a constant competitive ratio for the CNN problem *for any* $\lambda > 1$ ($\lambda = 3$ seems a good candidate); we also conjecture that the generalized Work Function Algorithm is competitive for the general 2-server problem.

The offline CNN problem seems interesting both in its own right and as a stepping stone for the online problem. More precisely, we want to find simple and fast memory-limited algorithms (exact or approximation) for the offline CNN problem. While a simple dynamic programming algorithm can be used to calculate optimal offline solutions, its state required grows with the length of the request sequence, and its time grows with the length squared.

For the Euclidean (planar) 2-server problem under the L_∞ norm, the intuition behind DC-TREE suggests that there is an algorithm with ratio better than 4 (which follows from the fact that 2 servers are 2 competitive), though it must be at least 3 (by Lemma 4). Any ratio under 4 would be interesting, especially by a simple algorithm.

We believe that for all three problems (sum of server problems, weighted k-server, L_∞ variant of the k-server problem), a generalized Work Function Algorithm has a competitive ratio at most a constant multiple larger than the optimal ratio.

We thank the anonymous referees for their helpful, in-depth comments.

References

[1] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 1999.

[2] Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. Searching with uncertainty. In *Proc. 1st Scandinavian Workshop on Algorithm Theory*, volume 318 of *Lecture Notes in Computer Science*, pages 176–189. Springer, 1988.

[3] Yair Bartal and Edward F. Grove. The harmonic k-server algorithm is competitive. *Journal of the ACM*, 47(1):1–15, 2000.

[4] Jon L. Bentley and Catherine C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Communications of the ACM*, 28(4):404–411, 1985.

[5] Dina Bitton and Jim Gray. Disk shadowing. In *Proc. 14th International Conference on Very Large Data Bases*, pages 331–338, 1988.

[6] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[7] Allan Borodin, Nathan Linial, and Michael Saks. An optimal online algorithm for metrical task system. *Journal of the ACM*, 39:745–763, 1992.

[8] Marek Chrobak, Howard Karloff, Tom H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

[9] Marek Chrobak and Lawrence L. Larmore. A new approach to the server problem. *SIAM Journal on Discrete Mathematics*, 4(3):323–328, 1991.

[10] Marek Chrobak and Lawrence L. Larmore. An optimal online algorithm for k servers on trees. *SIAM Journal on Computing*, 20(1):144–148, 1991.

[11] Marek Chrobak and Jiří Sgall. The weighted 2-server problem. In *17th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 593–604. Springer-Verlag, 2000.

[12] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. *Journal Computer Systems Science*, 48:410–428, 1994.

[13] Amos Fiat and Moty Ricklin. Competitive algorithms for the weighted server problem. *Theoretical Computer Science*, 130(1):85–99, 1994.

[14] Edward F. Grove. The harmonic k-server algorithm is competitive. In *Proc. 23rd Symposium on Theory of Computing*, pages 260–266, 1991.

[15] Anna Karlin, Mark Manasse, Larry Rudolph, and Daniel Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

[16] Elias Koutsoupias and Christos Papadimitriou. On the k-server conjecture. In *Proc. 26th Symposium on Theory of Computing*, pages 507–511, 1994.

[17] Elias Koutsoupias and David Scot Taylor. The CNN problem and other k-server variants. In *17th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 581–592. Springer-Verlag, 2000.

[18] Mark Manasse, Lyle A. McGeoch, and Daniel Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

[19] Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.

[20] David A. Patterson, Garth A. Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proc. of the 1988 ACM SIGMOD International Conference on Management of Data*, pages 109–116. ACM Press, 1988.

[21] René Sitters, Leen Stougie, and Willem de Paepe. A competitive algorithm for the general 2-server problem. In *Proc. 30th International Colloquium on Automata, Languages, and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 624–636. Springer-Verlag, 2003.

[22] Daniel Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.