The CNN Problem and Other k-Server Variants

Elias Koutsoupias[*] and David Scot Taylor[**]

University of California at Los Angeles, Los Angeles, CA 90095, USA
{elias, dstaylor}@cs.ucla.edu

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; 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 on-line 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, we show that no natural lazy memoryless randomized algorithm can be competitive.

The CNN problem also motivates 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 $\mathcal{L}_\infty$ norm for movement costs. We give a $\frac{1}{2}k(k+1)$ upper bound for the competitive ratio on trees.

1 Introduction

Consider a CNN crew trying to shoot scenes in 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 on-line, 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

[*] Supported in part by NSF grant CCR-9521606
[**] Supported in part by the Department of Energy under Contract W-7405-ENG-36

by the server. In the on-line version of the problem the requests are revealed progressively.

A more interesting formulation of the CNN problem results by assuming that we have 2 servers, moving in different dimensions, independent from each other. This allows us to generalize the problem as follows: there are two metric spaces M_1 and M_2 with one server in each one. 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.

More generally, let $\tau_1, \tau_2, \ldots, \tau_n$ be task systems [7] (not necessarily distinct). We can synthesize these task systems to get two new interesting on-line problems: the *sum* and the *product* of $\tau_1, \tau_2, \ldots$. The sum is the problem where we get requests (tasks) for each task system and we have to service only one of them. The product, on the other hand, is the problem where we have to service all of them. When all task systems are identical, $\tau_i \equiv \tau$, the product is related to randomized on-line algorithms for the task system τ. It is a trivial fact that a deterministic algorithm for the product of n copies of τ, with each request the same across all spaces, is not different than 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. In fact, it is one of the simplest conceivable sum problems and a stepping stone towards building a robust (and less ad hoc) theory of on-line computation. Despite its importance, there is no definite positive result for this problem. The lack of results should be attributed to the hardness of the problem rather to the lack of interest. The problem has been known to the research community for quite a few years[1]. The CNN problem and more generally the sum of on-line problems also give flexibility to model problems which the k-server problem cannot. 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, 4, 16, 20, 21]. Each disk may have information in completely different locations, leading to independent costs for information retrieval. We wish to minimize time spent looking for data, but do not actually care which disk the information comes from. In contrast, writing must be performed to all disks; this is closer in spirit to the product on-line problem mentioned above, but in our worst case analysis, there will be little or no writing at all.

In this work, we use *competitive analysis* [5, 17, 24] to evaluate the quality of on-line algorithms; the competitive ratio is defined to be the worst-case performance of the on-line algorithm, compared to the optimal cost for the same sequence of requests. More precisely, algorithm ALG is c-competitive if there is a

[1] 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.

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 on-line 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 no known upper bounds. Compare this with the k-server problem: although the k-conjecture has not been resolved yet, we now know the competitive ratio within a factor of 2 [18]. In particular, the 2-server problem was settled from the beginning [19]. 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 prove that there is no competitive natural lazy memoryless randomized algorithm, (Theorem 1) for the CNN problem. We also 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 servers have different speeds and the cost is proportional to the time needed to service a request. We show (Theorem 2) that any deterministic on-line algorithm has a competitive ratio at least $6 + \sqrt{17}$ for the weighted 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 are not competitive for the CNN problem (Proposition 1). Some of the results extend directly to the CNN problem in higher dimensions (the sum of more 1-server problems), in which the lower bounds of [13] also apply. It is easy to show that for the sum of any k non-trivial spaces (at least 2 points each) a $2^k - 1$ lower bound on the ratio exists, and that for the simplest spaces (k spaces of 2 points each, 1 unit apart), this bound is tight.

Finally, we study a variant of the k-server problem, motivated by the CNN problem. Instead of trying to minimize the cost of moving servers, we try to minimize the *time* spent waiting for service, but we allow multiple servers to move simultaneously [12]. When a request is made, the on-line algorithm specifies possible movement for each of the k-servers, and tries to minimize the cost of their total movement, under the $\mathcal{L}_\infty$ norm. For k-servers in a tree, we determine the exact ratio $\frac{1}{2}k(k+1)$ of the DC-TREE algorithm of [8, 9]. In particular for $k = 2$, we show that DC-TREE is optimal with competitive ratio 3.

2 Memoryless Randomized Algorithms

Our first result concerns randomized algorithms for the CNN problem. Within the plane, we define a *natural* memoryless algorithm to be an algorithm which has the following property: *The probability of satisfying any request by moving the shorter distance is only a function of the ratio of the shorter distance to the longer distance.* We break this into several subproperties:

- Translational Invariance: The algorithm does not consider what point its server is at, only relational positions of points within the space (such as the distances to the request).
- Symmetry 1: The algorithm is unbiased towards horizontal and vertical moves. More precisely, let (x, y) be the on-line position and consider two possible next requests at $(x + a, y + b)$ and $(x + b, y + a)$. If the server moves horizontally with probability p to service the first request, it moves vertically with the same probability p to service the second request. Similarly, request $(x - a, y + b)$ also moves horizontally with probability p.
- Scalability 1: Let (x, y) be the on-line position and consider two possible next requests at $(x + a, y + b)$ and $(x + \lambda a, y + \lambda b)$, for some λ. The server moves horizontally with the same probability for both requests.

We call these properties natural because they reflect the fact that for this problem, the plane is symmetric over $y = x$, $y = 0$, and $x = 0$, and is scale and translation invariant. We also require that the algorithm be *lazy*: if the algorithm is at position (x, y), and the request is at (r_x, r_y), it will service the request by moving to either (r_x, y) or (x, r_y). It is standard to require memoryless algorithms to be lazy, otherwise the on-line algorithm can try to encode history information within its non-lazy movement. Although this encoding is impossible for natural algorithms, we nevertheless require the laziness property.

Unlike the k-server problem for which a natural lazy memoryless algorithm, HARMONIC, has finite competitive ratio ($O(2^k \log k)$, [3, 15]), no such algorithm for the CNN problem is competitive.

Theorem 1. *There is no competitive natural lazy memoryless randomized algorithm for the* CNN *problem.*

Proof. Fix some memoryless algorithm with optimal competitive ratio. Assume that the adversary is at position $(0, 0)$ (by Translational Invariance). We want to estimate the on-line cost when the adversary produces a worst-case sequence of *lazy requests*, defined as requests for which it does not have to move, i.e., points from the lines $x = 0$ and $y = 0$. Let $\Phi(x, y)$ be the expected cost of the on-line algorithm to converge to $(0, 0)$ starting at position (x, y).

It follows for the above properties of the on-line algorithm that Φ satisfies:

- Symmetry 2: $\Phi(x, y) = \Phi(|y|, |x|)$.
- Scalability 2: $\Phi(\lambda \cdot x, \lambda \cdot y) = \lambda \cdot \Phi(x, y)$.

To prove our theorem, we examine various possible combinations of on-line positions and requests. We start by considering $\Phi(1, \varphi)$, where $\varphi = \frac{1+\sqrt{5}}{2}$, the golden ratio. We try to find recursive bounds on Φ. The following table summarizes combinations on-line positions, requests, and resulting bounds on Φ.

On-Line Position	Next Request	Resulting Inequality	Justification
$(1,\varphi)$	$(\varphi^2,0)$	$\Phi(1,\varphi)$ $\geq \frac{1}{2}(\Phi(\varphi^2,\varphi)+\varphi^2-1)+\frac{1}{2}(\Phi(1,0)+\varphi)$ $=\frac{1}{2}(\varphi\cdot\Phi(\varphi,1)+\varphi)+\frac{1}{2}(\Phi(1,0)+\varphi)$ $=\frac{\varphi}{2}\Phi(1,\varphi)+\frac{1}{2}\Phi(1,0)+\varphi$	Symmetry 1, $\varphi^2-1=\varphi$ Scalability 2, $\varphi^2-1=\varphi$ Symmetry 2
$(1,0)$	$(0,1)$	$\Phi(1,0)$ $\geq \frac{1}{2}(\Phi(1,1)+1)+\frac{1}{2}(\Phi(0,0)+1)$ $=\frac{1}{2}\Phi(1,1)+1$	Symmetry 1
$(1,1)$	$(0,\varphi)$	$\Phi(1,1)$ $\geq p(\Phi(1,\varphi)+\varphi-1)+(1-p)(\Phi(0,1)+1)$ $=p(\Phi(1,\varphi)+\varphi-1)+(1-p)(\Phi(1,0)+1)$	For some p. Symmetry 2
$(1,0)$	$(0,\varphi)$	$\Phi(1,0)$ $\geq p(\Phi(0,0)+1)+(1-p)(\Phi(1,\varphi)+\varphi)$ $=(1-p)\Phi(1,\varphi)+p+(1-p)\varphi$	Symmetry 1, Scalability 1

If we assume that $\Phi(1,\varphi)$ is bounded we get a contradiction: from the first three bounds we get $p<\varphi-1$ and from the first and last $p>\varphi-1$. Therefore, if the initial position is $(0,0)$ and the adversary moves to position $(1,\varphi)$, the on-line cost to converge to $(1,\varphi)$ is unbounded. This shows that the competitive ratio is unbounded. □

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[2] and we wish 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 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 "simulate" the BRIDGE (or cow-path) problem [2, 23]. Thus, the CNN problem contains as a subproblem another fundamental on-line problem. Interestingly, we know of two different ways to view the bridge problem as a special case of the CNN problem. This shows the close connection between these two problems, although the CNN problem is, of course, much more complicated.

The BRIDGE problem is a simple on-line 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).

Theorem 2. *For the 2-server problem in a line, in which one server is m times faster than the other, the deterministic competitive ratio is at least $6+\sqrt{17}$. More precisely, for any $\epsilon > 0$, there is a sufficiently large m such that the competitive ratio is at least $6 + \sqrt{17} - \epsilon$.*

Proof. 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.

[2] To make this interpretation strict, we consider that we are only allowed to move one server at a time, which 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 Chebychev distance ($\mathcal{L}_\infty$) instead of the Manhattan distance ($\mathcal{L}_1$). 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 4.

We shall assume $m >> 1$. Also, without loss of generality, we can assume that the on-line 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 on-line slow server is at r the adversary's strategy distinguishes two cases: if 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. The adversary's strategy when the slow server is at l is symmetric.

This adversary's strategy forces the slow server to explore larger and larger portion of the line. The slow server at r cannot continually move to the right, because this would result in competitive ratio m. Eventually, to achieve ratio less than m, the slow server will need to go to point l. The adversary can continue to force the slow server to "zig-zag", exploring larger and larger segments of the line; exactly as the explorer does in the BRIDGE problem.

Assume that at the end of the game, the slow server has explored the interval $[-z, y]$, for $z \geq y$, and 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. (To be fair, m should be fixed before the on-line algorithm is forced to choose z values, but we can always guarantee that z is no more than some modest constant multiple of y or else we again reach ratio m.)

Thus, the competitive ratio is at least $9 - \epsilon$, where ϵ tends to 0 as m tends to ∞. To simplify the presentation, we henceforth will drop the ϵ term.

It takes a little effort to improve the bound to 10. Observe that we ignored the cost of the on-line fast server. Let x_0 be this cost. The adversary has the alternative strategy to service all requests with its fast server. Its total cost then is x_0. Thus, the competitive ratio is at least

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

The minimum value, 10, is clearly obtained when $x_0 = y$.

To get the improved lower bound $6 + \sqrt{17}$, we extend the above sequence of requests. Let ρ be the sequence of requests described above (which completes the "BRIDGE simulation") and 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, the on-line 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. Let j_i be the number of repetitions of $(0, -z)$ needed to make the on-line 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 on-line slow server move back to y for the i^{th} time. Let $x_{2i-1} = j_i z/m$, the movement of the on-line fast server before the i^{th} move to 0, and $x_{2i} = k_i z/m$, the movement of the on-line fast server before the i^{th} move back to position y.

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

$$\max_{x_i,y}\left(\frac{9y+x_0}{y}, \frac{10y+x_0+x_1}{x_0}, \frac{11y+x_0+x_1+x_2}{y+x_1}, \frac{12y+x_0+x_1+x_2+x_3}{x_0+x_2}, \ldots\right),$$

and in general:

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

The different denominator types are from the two off-line strategies we have already seen for the ρ requests: moving both servers, or just the fast one. The adversary can choose either of these two[3].

It turns out that to minimize the expression above, we can assume that 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:

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

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

$$\forall i \geq 2,\ x_i = (x_0+8)^{i-1}x_1 - 2\frac{(x_0+8)^{i-1}}{x_0+7} + \frac{2}{x_0+7}$$

and finally

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

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 $6+\sqrt{17}$. □

[3] In fact, the off-line servers have a third main strategy to consider, which is moving the slow server to point z and servicing the rest of the queries with the fast server. By eliminating this option, we may be proving a slightly *weaker* lower bound than possible from these three strategies; we can show however that the third strategy does not improve the lower bound by more than 0.22. The equations are complicated by the fact that once the third option is added, it is no longer safe to assume that the optimal BRIDGE solution for ρ will give the best solution to the equations.

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.

Proposition 1. *For* 2*-servers in a line, one with speed* 1 *and the other* m*, no* DC *or* BAL *type algorithm has constant competitive ratio (bounded independently of* m*).*

Obviously these algorithms need to be modified 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 expect that the statement holds for any variant which maintains the spirit of either algorithm, and use a simple example as an intuitive justification. For DC, consider an on-line configuration at $(0, x)$, where the fast server is at 0, and x large enough that the slow server will reach a request at $x+1$ before the fast one ($(x+1)/m > 1$ will do for the most natural generalization of DC) By repeating the sequence of requests $0, x, 0, x+1$, the on-line servers will pay cost 2 for every 4 requests, while an adversary could satisfy them with just cost $2/m$ (assume $m \geq 1$). By making m large, we can get an arbitrarily large competitive ratio. A similar example can be used for BAL.

Leaving our k-server interpretation behind, we can achieve the same results for the CNN problem 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. Now any request can be satisfied at a cost of 1 by moving from one line to the other. We use a strategy similar to before. 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 $(0, r+1)$. We use these requests to again make the server move away from the center just as before, simulating the BRIDGE explorer, and vertical movement corresponds to movement of the fast server in the previous argument. At the end, the adversary can again make the server alternate between the origin and the shorter "arm" of exploration. All equations are the same.

4 The k-Server Problem under the $\mathcal{L}_\infty$ Norm

In this section, we consider the k-server problem, where servers have the same speed, but can be moved simultaneously, and *time* of service is to be minimized.

Once a request is served, the next request is given. By ignoring the fact that our algorithm (and the optimal one) can move more than one server simultaneously, we can achieve a competitive ratio for this problem which is k times larger than for the regular k-server problem. Using the best known bound of [18], this gives a $2k^2 - k$ bound for this ratio, but we expect that this can be improved. For example, within the uniform metric space, 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 [9], 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 $\mathcal{L}_\infty$ norm,* DC-TREE *has competitive ratio $\frac{1}{2}k(k+1)$.*

Proof. Let M_{min} be the distance for the best matching between the on-line and off-line servers, and ΣDC-TREE be the distance between all pairs of on-line servers. To show that DC-TREE is competitive, consider some phase in which a fixed number of servers are moving, and use the following potential:

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

The analysis is as in [9]. For a move of cost d, the off-line server can increase M_{min} by kd, giving a ratio of $\frac{1}{2}k(k+1)$.

To show this ratio is tight for DC-TREE, consider servers (on-line and off-line) at position $(2, 4, \ldots, 2k)$ of a line. For a cost of 1, the off-line 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 off-line position, at which time we are at a position similar to the original one. □

The following lemma shows that the DC-TREE algorithm is optimal for $k = 2$ and Euclidean metric spaces.

Lemma 1. *No on-line algorithm has competitive ratio less than 3 for the Euclidean 2-server problem under the L_∞ norm.*

Proof. It suffices to consider lazy on-line algorithms. Consider requests in a line, with initial configuration (on-line and off-line) $\{0, 2\}$. If there is a request at 1, by symmetry, the on-line 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. Now the adversary requests point 3 (and reveals its configuration $\{1, 3\}$). The on-line algorithm must pay at least 2 to service the request. The total on-line cost is 3, but the off-line 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 on-line algorithm has ratio less than 3. □

In [11], the DC-TREE algorithm for $k = 2$ can be extended to any metric space, not just trees. This is done by considering an extension of the metric space to its "closure" so that any three points are connected by a tree. Consider "virtual" movement through the closure of the space, and only actually move a server when it needs to service a request, which will be a point in the original space. By the triangle inequality, the actual movement will be less than the total virtual movement. Unfortunately, under the $\mathcal{L}_\infty$ norm, this strategy does not work: except for the last move (to the original space), the virtual movement is for free, with the cost for the move being dominated by the movement of the other server. When it comes time to move the server to answer a request, this virtual movement is no longer available for free. Triangle inequality can no longer guarantee that the actual movement is less than the sum of virtual movement.

5 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 few of them here.

The CNN problem is wide open. We conjecture that it has a finite competitive ratio. In fact, we conjecture that the generalized Work Function Algorithm (the one that moves a server which minimizes $\lambda w(A') + d(A, A')$) has constant competitive ratio for the CNN problem for any $\lambda > 1$ ($\lambda = 3$ seems a good candidate); we conjecture the same for the sum of two 1-server problems in general.

The off-line CNN problem seems interesting in its own right and as a stepping stone for the on-line problem. More precisely, we want to find simple and fast approximation algorithms for the off-line CNN problem.

For the Euclidean k-server problem under the $\mathcal{L}_\infty$ norm, the intuition behind DC-TREE suggests that there be 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 1).

We believe that for all three problems (sum of server problems, weighted k-server, $\mathcal{L}_\infty$ variant of the k-server problem) the Work Function Algorithm has almost optimal competitive ratio. We don't have a candidate randomized algorithm though.

References

1. Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. In *Proc. 26th Annual ACM Symposium on the Theory of Computing, (STOC '94)*, pages 593–602, 1994.
2. R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins. Searching with uncertainty. In *1st Scandinavian Workshop on Algorithm Theory, (SWAT 88)* pages 176–189. Springer-Verlag, Lecture Notes in Computer Science 318, 1988.
3. Y. Bartal and E. F. Grove. The harmonic k-server algorithm is competitive. Unpublished Manuscript, 1994.

4. D. Bitton and J. Gray. Disk Shadowing. In *Proceedings of the 14th International Conference on Very Large Data Bases, (VLDB '88)*, pages 331–338, 1988.
5. J. L. Bentley and C. C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. In *Communications of the ACM, 28(4)* pages 404–411, April 1985.
6. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge Mass., 1998.
7. A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, (STOC '87)* , pages 373–382, 1987.
8. M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. In *SIAM Journal on Discrete Mathematics*, 4(3):323–328, May 1991.
9. M. Chrobak and L. Larmore. An optimal on-line algorithm for k-servers on trees. In *SIAM Journal on Computing*, 20(1):144–148, February 1991.
10. M. Chrobak and L. Larmore. The server problem and on-line games. In *On-Line Algorithms, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 7, pages 11–64, 1991.
11. M. Chrobak and L. Larmore. A new approach to the server problem. In *SIAM Journal on Discrete Mathematics*, 4(3):323–328, 1991.
12. A. Fiat, Y. Rabini, and Y. Ravid. Competitive k-server algorithms. In *31st IEEE Annual Symposium on Foundations of Computer Science, (FOCS '90)*, pages 454–463, October 1990.
13. A. Fiat and M. Ricklin. Competitive algorithms for the weighted server problem. In *Theoretical Computer Science, 130(1)* pages 85–99, August 1994.
14. S. Gal. *Search Games*, Academic Press, 1980.
15. E. F. Grove. The harmonic online k-server algorithm is competitive. In *Proc. 23rd Symposium on Theory of Computing, (STOC '91)*, pages 260–266, 1991.
16. F. Kurzweil. Small disk arrays - the emerging approach to high performance. Presentation at COMPCON 88, March 1, 1988.
17. A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. In *27th Annual Symposium on Foundations of Computer Science, (FOCS '86)*, pages 244–254, October 1986.
18. E. Koutsoupias and C. Papadimitriou. On the k-server conjecture. In *Proc. 26th Symposium on Theory of Computing, (STOC '94)*, pages 507–511, 1994.
19. M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for on-line problems. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, (STOC '88)*, pages 322–333, May 1988.
20. R. Muntz, J. R. Santos, and S. Berson. A parallel disk storage system for real-time multimedia applications. In *International Journal of Intelligent Systems, Special Issue on Multimedia Computing System*, v.13, n.12, pages 1137–74, December 1998.
21. D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). In *ACM SIGMOD Conference Proceedings*, pages 109–116, 1987.
22. C. Papadimitriou and M. Yannakakis. Shortest Paths Without a Map. In *Proc. 16th Internat. Colloq. Automata Lang. Program.*, vol. 372 of *Lecture Notes in Computer Science*, pages 284–296. Springer-Verlag, 1989.
23. C. Papadimitriou and M. Yannakakis. Shortest Paths Without a Map. In *Theoretical Computer Science, 84(1):127–150*, 1991.
24. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. In *Communications of the ACM, 28(2):202–208*, February 1985.