Advice Complexity of CNN Problem

Sameer Naib
COMP 7720
Department of Computer Science
University of Manitoba, Canada
naibs@myumanitoba.ca

Manmohit Singh
COMP 7720
Department of Computer Science
University of Manitoba, Canada
singhm30@myumanitoba

Abstract

The advice complexity of online algorithms has improved the performance of online algorithms by analyzing the problem and the input data and feeding the algorithms with some useful information based on the analysis to make some right decisions. A significant amount of research has been done on ‘k-server’ problem with advice. The main objective of this paper is to improve the competitiveness of an important variant of online k-server problem called the CNN problem by introducing the concept of advice. The paper discusses the lower and upper bounds of the size of advice bits required to achieve an optimal solution in various types of CNN problem including 1 server and multi-server variant.

Introduction

The step by step method of solving a problem in finite amount of space and time is called an algorithm. A significant amount of research has been done on the offline algorithms in which the input data is known to the designer in advance and the objective is to get an output with minimum time and space complexity. An online algorithm in contrast is a realistic version of analyzing problems in which the input sequence is fed to the algorithm in a sequential manner such that every new request arrives after the algorithm has processed the previous request. A formal definition of online algorithm is given below [1, 2].

Definition 1: Online Algorithm: *An online algorithm ‘A’ processes an input sequence X = {x_1, x_2, x_3 ...x_n} piece by piece such that the input x_{i+1} is revealed to ‘A’ after it has processed the input x_i and cannot change its previous decisions after looking at the future requests.*

In distinction from the offline algorithms, the performance of an online algorithm is not measured in terms of time and space. The competitiveness of an online algorithm is measured by comparing its worst case performance against the optimal (best) offline algorithm which knows the future sequence in advance. An online algorithm is said to be c-competitive if for a sequence X, the cost of algorithm, cost (Alg(X)) $\leq$ c*cost (Optimal) + α, where $c \geq 1$ and α is some constant value [3].

The online approach has immense applications in path problems, list update problems, graph-coloring and clustering. "**k-server**" problem has been one of the most prominent problems in the field of online algorithms. This problem deals with allocating servers to various incoming points on different locations on the metric space in an efficient manner. Efficiency is measured in terms of distance travelled by the servers to serve all incoming requests relative to the optimal offline adversary who unlike online algorithm knows future requests and takes right decisions and paths [4, 5].

The CNN problem - A variant of the 'k' server problem is the CNN problem on a plane, introduced by Koutsoupias and Taylor [6]. This problem is different from the k-server in a sense that, the server need not to move to the exact same position of the request, it can just reach the same horizontal or vertical line of the request to serve it. It was named after popular news channel CNN whose crew was given a task of streaming relevant events in the various locations of Manhattan such that their powerful cameras had capability of capturing an event by being on any position of the cross streets. The formal definition is discussed further. For the implementation purposes consider the cameras as servers and the occurring incidents as the input requests. Hence for handling a request, the server only needs to be either horizontally or vertically aligned with the input request instead of traversing all the way to the exact location of that request. There can be multiple variants of the CNN problem that includes 1 server, multi-server and orthogonal CNN problem. Formally, a 1 server CNN problem can be defined as follows:

Definition 2: 1-server CNN Problem: *Consider a server at any position on a plane with request coming online at different positions of the metric. A server serves a request at position (x_1, y_1) by either moving to a vertical line $x = x_1$ or by moving to horizontal line $y = y_1$.*

Further Extension of the CNN problem is the **orthogonal CNN problem** proposed by Iwama and Kouki [7] which has a restriction that every new request should be either x-aligned or y-aligned with the previous request. Formally, a problem is said to be orthogonal only when the new request r_{i+1} has either same 'x' co-ordinate or the 'y' co-ordinate as that of a previous request r_i.

Online Algorithms with advice – Since the online algorithms do not have any clue about the incoming requests and moreover they cannot change their decisions once made, they generally end up with having difficulty in matching up with the performance of offline algorithms. In order to help online algorithms in reaching optimality as the offline algorithm, advice was proposed. Helping online algorithms with advice was proposed in 2009 [2]. The objective of this method is to analyse the problem and input data and evaluate the information that can be useful while processing the input [8]. This information is encoded in the form of binary bits that has two values '0' or '1'. There has been various models being introduced to handle the advice bits but the model given by Bockenhauer et.al. called the tape model has been proved quite efficient [9]. The binary bits are stored on an advice tape of an infinite length by an oracle who has access a previous access to the input data and this advice is fed serially with the input data in online manner.

The question is that how many advice bits an online algorithm needs to be more competitive? The advice complexity of an algorithm is defined as the maximum number of advice bits required to get an optimal solution. ‘k-server’ has been one of the first problems which have been worked with advice following up with many other problems [3, 10, and 11].

Finding the advice complexity of the CNN problem is the core objective of this paper. This paper deals with finding the relevant information for the CNN problems and its variants which can be useful for the online algorithms while making their decisions in serving the requests by servers. The idea is to compute the upper bound and the lower bound on the advice size for an input sequence of length n, n being quite large. In the initial part of the methodology, we have computed an upper bound for the advice size for 1 server and multi-server CNN problem which is *O*(n) and *O*(n*log k) respectively, ‘n’ being the length of the input sequence and ‘k’ being the number of the severs. This is followed up with the calculation of the lower bound of 1 server problem using binary guessing approach which shows that the bound on advice size is tight. Then in the later part, we have analyzed the performance of the online algorithm when advice size is less than *O*(n).

This report is further divided into three major sections. In the first section, we discuss some online algorithms being proposed for the CNN problem and its variants. The second section deals with computing the advice complexity of the CNN problem and the last section summarizes the results and highlights some crucial parts to be worked in future.

Background Information

The CNN problems have been studied since late 1990‟s, formally introduced by [6]. They found that the competitive ratio of any deterministic online algorithm with discrete requests is no less than $6 + \sqrt{13}$. An algorithm was introduced in [12] in which upper bound of the problem was proved to be 10^5 which was later improved to 879. Major improvement was seen with the introduction of orthogonal CNN variant with a competitive ratio of at most 9 [7].

Furthermore Augustine and Gravin [13] worked with the continuous CNN problem in which, the request can move continuously and the server must continuously serve the incoming requests. They proposed an algorithm that used knight moves for moving the server i.e. either 2 moves horizontally along with 1 move vertically or 1 move horizontally followed by 2 moves in vertical direction. Using this approach, they proved a lower bound of 3 along with the upper bound $3 + 2\sqrt{3} \approx 6.464$. Their method can be generalized to discrete orthogonal CNN problem, so they got an improved competitive ratio over 9 in the previous case.

Advice in the case of CNN problem is a novel topic and requires plenty of work to be done for studying that how advice of various sizes can improve the performance of the algorithms. As mentioned earlier, advice was initially used in k-server problem. It was followed with implementation in various online problems like online bin packing, list update problems and online clustering [3, 10, 11, and 15].

Methodologies

In this section, we investigate the upper and lower bounds on the number of advice bits required to achieve an optimal solution for the CNN problem. The first part describes the upper bound for both 1 server and Multiserver CNN problem. It shows the maximum number of bits required to be as good as an optimal algorithm which knows the location of future requests.

The second section deals with the lower bound on the advice size for the special case of Orthogonal CNN problem. The lower bound is proved with help of Binary Guessing problem which will be discussed as well. The last section discuss the improvement in performance of online algorithm when advice size is less than the proved upper bound.

I. Upper Bound

For an online algorithm to achieve an optimal solution, it requires to read some advice about the input data or decisions that an offline oracle makes during its execution. There are some questions that need to be answered during inspection of the advice for an online algorithm. These questions are:

1. What does the advice encode?
2. What should be the size of advice?
3. How does advice help?
4. How does online algorithm reach optimality with help of advice? [15]

Before explaining the advice complexity, we would like to discuss the lazy approach of an optimal offline algorithm. The concept of lazy algorithms was introduced in late 90's with respect to k-server problem [1].

An algorithm is said to be lazy if it allows only 1 server to move while serving the request to any node in the k-server problem or in our context, the CNN problem. The basic objective behind the lazy approach is that it minimizes the distance moved by the servers to serve the request. For any incoming request, an offline algorithm which knows the future request and has power to change its decisions, move multiple servers to reach the requested node. These multiple servers are called virtual servers. Moreover, in case of multiple paths and directions in the CNN problem, the virtual servers move in both horizontal and vertical direction to reach the axis aligned to the requested node. The distances of multiple servers in different directions are analyzed and one with the minimum distance is chosen as the real server. In case of 1 server problem, a single server moves in multiple directions to serve the request. The minimum distance from the virtual paths are analyzed and the real server moves to the requested node.

The optimal algorithm does not follow the greedy approach. In a general way, we can say that any logical algorithm which processes the request analyzes the vertical and horizontal distance of the request node from its position and chose the path with the minimum distance. Whereas, an optimal algorithm may or may not follow this approach and sometimes it takes a costly move initially which proves to be a right decision in the long run as it knows the pattern of input sequence in advance. Consider the following example.

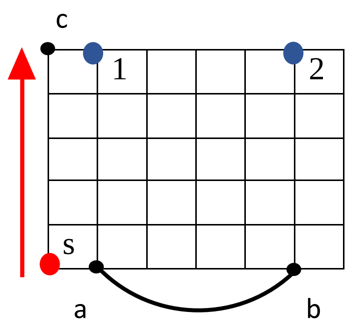

Fig1. Greedy is not optimal

In figure 1, there is a 1 server CNN problem. The input sequence is $(1, 2)^n$ where n is quiet large. Any logical algorithm which motivates to move less will move server 1 at position a as it is at a distance of 1 unit from the server's position. For request to 2, it moves to position b horizontally which is at distance of 4 units as compare to 5 units while making a vertical move. This continuous and the method pays a cost of $1+ 4^n$**.** Whereas the optimal algorithm which knows the future sequence, will move to position c which is at a distance of 5 units and pays no cost further. Thus we can say that in this case, competitive ratio of the algorithm will be $\frac{Cost(Alg)}{Cost(Opt)} = \frac{1+4n}{5}$ which is equivalent to **n**. So, we can say that unlike the algorithm which operated in a greedy manner, an optimal algorithm makes right decision to serve any request by moving to any axis aligned with the node. Moreover, we can see that the example showed one of the worst case scenarios as optimal algorithm just took 1 move as compare to the online algorithm which had to move server consecutively for every request. We can generalize the following theorem:

Theorem 1

In 1 server CNN problem, any deterministic online algorithm that tends to compare the horizontal and vertical distance of the request from server's position and takes the minimal path, has a competitive ratio of at most n where n is the length of the input sequence.

The minimum path between a point where a server is located to one of the axis aligned with a requested node is always a straight line that may a horizontal line or vertical line. The proof is quite simple. We are all aware of the mathematical theorem.

Theorem 2

The shortest distance between a point and a line on a plane is equal to the perpendicular distance between the line and the point.

The figure 2 shows an example of serving 2 request nodes A and B by server S. For A, server S can reach both bold axis aligned to A. It decides to take horizontal direction which is closer. For B, vertical move is preferred as it is shorter than horizontal move.

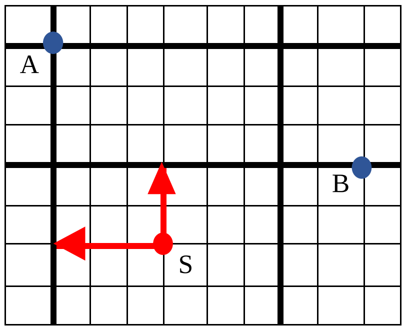

Fig 2. Server moves in only 1 direction

So from theorem 2, it is clear that minimum distance between the server and the axis aligned to the new requested node will be a straight line on the grid. Therefore, we can say that the optimal lazy algorithm which moves only 1 server will move only in 1 direction which may be horizontal or vertical depending on the future sequence. We have lemma here.

Lemma 1

The optimal lazy algorithm moves only 1 server in 1 direction to serve any request.

The upper bound is defined for both 1 server and Multiserver CNN problem. 1 server is discussed first. As there is only 1 server to serve all requests, so optimal lazy algorithm analyzes all non-lazy paths traversed by the virtual server. The minimum path travelled is set as the real server path. From lemma 1, it is proved that the optimal server will move either horizontally or vertically. So, the question arises that what should be the advice to be as good as an optimal algorithm.

For serving any request, online algorithm must know the decision made by the optimal algorithm for that particular request. We have seen that optimal algorithm makes a horizontal move or a vertical move. So, the advice for any request must encode whether optimal algorithm made a horizontal move or a vertical move. This can be encoded with 1 binary bit of advice per request. '1' can denote a horizontal move and similarly '0' can indicate a vertical move. Suppose there are n requests out of which m requests require a move of a server. This means that n-m requests were automatically hit with the previous position of the server. So at most 'm' bits of advice is required for 'm' requests which is equivalent to ***O*(n)**. The online algorithm reads the advice bit for each request and makes the corresponding move to serve the request. We can say that the method will be optimal as online algorithm will make same decisions which optimal algorithm would have made in the same case. Therefore, in order to achieve an optimal solution, linear size of advice (*O*(n)) is required. So, from the above results, we have the following theorem:

Theorem 3

For a 1-server CNN problem, an advice of at most O(n) is required to achieve an optimal solution where 'n' is the length of input sequence.

For Multiserver CNN problem, advice must indicate the optimal server as well as the direction of move to achieve an optimal solution. We know that the optimal algorithm will follow lazy approach and only 1 server moves and make a horizontal or vertical move. If 'k' is the number of servers then any 1 server can be represented by 'log k' number of binary bits. Considering the same situation of m number of requests in which online algorithm has to move any server, then (m*log k) number of bits are required to denote the server which moves along with 'm' number of bits to give the direction of move. So in total, at most (m*log k + m) number of bits are required which is equivalent to ***O*(n log k).** For each request which is not a hit, online algorithm reads the advice bits (log k) which tells the server that optimal algorithm moves and 1 bit which indicates the direction of move. We can say that online algorithm just mimics the optimal offline algorithm as it makes exact same decisions as optimal algorithm. Therefore, with Multiserver CNN problem as well, an advice of linear size (*O*(n log k)) is required. From the above results, we conclude the following theorem:

Theorem 4

For a Multi-server CNN problem, an advice of at most O(n log k) is required to achieve an optimal solution where n is the length of *input sequence and k is the number of servers.*

II. Lower Bound

The lower bound is used to indicate the minimum number of advice bits required to achieve an optimal solution. We are going to analyze a special case of 1 server orthogonal CNN problem for the lower bound. We have seen that an advice of linear size (O(n)) is required to achieve an optimal solution. In order to devise lower bounds for this problem, we are going to check if an advice of sub linear function of n is sufficient to achieve optimality. The answer is No. This is proved with the help of **Binary String Guessing Problem**. Binary Guessing problem has been a powerful method to compute the lower bound on the number of advice bits necessary for an online algorithm to be as good as optimal offline algorithm [14]. This method has been used earlier for proving lower bounds of many online problems like k-server and online bin packing [10, 11].

Definition 3: *The binary string guessing problem is an online problem in which input is a sequence of binary bits of length n as (X = $x_1, x_2, x_3 \ldots x_n$). For any input x_i, the online algorithm must guess each input to be either '0' or '1'. After the guess has been made, the bit is revealed and the next guess is made [14].*

The objective is to correctly guess maximum number of bits. In case of adversarial generated sequence, all guesses can be made wrong as adversary knows the guess made by the algorithm and generates the opposite bit. All guesses can be made wrong by the adversary in the worst case scenario. Therefore, advice is needed to guess the bits right. 1 bit of advice can be sufficient to guess at least half of the bits. The advice must encode the bit out of '0' or '1' which occurs more number of times. After reading the advice bit, algorithm just guesses only the more frequent bit and makes at least half of the guesses right. But, in order to guess more than half number of the bits correctly, an advice of linear size is required. A formal definition of such idea is written below.

Lemma 2

"For a sequence of length n, any deterministic algorithm that wants to guess more αn bits correctly where $1/2 < \alpha < 1$, needs at least $(1 + (1 - \alpha) \log(1 - \alpha) + \alpha \log \alpha) * n$ bits of advice" [10].

In order to show that the bound on the advice size is tight in the case of 1 server orthogonal CNN problem, we reduce the problem to a simple instance of binary guessing problem which suffices to show that the an advice of linear function of n is required to attain a competitive ratio of 1. Consider a following 2 X 2 grid with server at position '6' with 2 types of input sequence A and B which occurs m times where m is arbitrary large value. Online algorithm does not know the type of sequence being requested. The input sequence follows the orthogonal property. The distance between two adjacent points is 1.

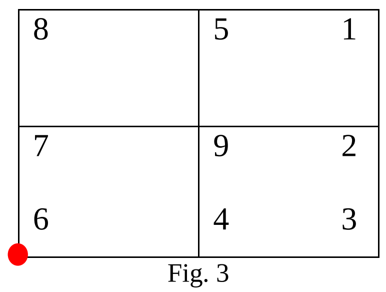

Fig. 3

Type A: 12345467**863686368**

Type B: 158727643**68636863**

Any reasonable algorithm will have server at position 6 at start of every phase. We can formalize it with following lemma.

Lemma 3

Any algorithm A can be modified to another algorithm B without increase in cost (may be decrease in some case) such that position of server at the start of every phase is at position 6.

This can be proved by taking an instance when server is not at position 6. We can see that the type A as well type B input sequence ends with alternating 8 requests to positions 6, 8 and 3 starting with request to position 6 as its 10th request. This means that the positions 7, 8, 4 and 3 are other positions than 6 that can serve request to 6 at this moment. Moreover, it implies that the maximum cost that any algorithm will pay by moving to position 6 will be 2 i.e. distance between 8 and 6 or 3 and 6 at this point. The cost of last 7 requests by moving to position 6 can be at most 2 as it is hit for other requests. On the other hand, if algorithm does not decide to move at position 6. In such case, algorithm has to pay more cost. Consider server is at position 7 or 8 for request to 6 for type A sequence. Then for request to 3, it has to pay at least a cost of 2 to reach position 4 so that it does not pay any cost for next request to 6. After serving 3, later it has to serve 8 and in this case it has to pay a cost of at least 2 to reach position 7. Again for subsequent requests to 8 and 3 it pays cost of at least 2 for both cases. So, algorithm pays a cost of at least 8 here. Now consider server is at position 4 or 3. In this case, server does not move for request to 3. But for subsequent 3 requests to 8 and 3, it pays cost of at least 2 for each case. It pays at least 6. Same is the case with Type 2 sequence. We can see that if server is at position 6 at 11th request, the server saves its moving distance and needs not to move for the rest of sequence. So, it is reasonable to have server at position 6 at the start of every phase.

Before analyzing the online algorithm approach, we can see an optimal offline approach which knows the type of sequence being requested. Both of the sequences have initial request to position 1. Server can move to position 8 or 3 with a minimum cost. The second request is to the position 2 in case of type A and position 5 in case of type B. For type A, moving the server from 6 to 3 is ideal for initial request to 1 as it automatically serves request to 2 and moreover subsequent 2 requests to 3 and 4. For next request to 5, server moves from 3 to 4 as the further request is to 4 and 6. For request to 7 server moves from 4 to 6 as the further requests are to 8 and 6. Then from the above lemma, we know that staying at position 6 is optimal for ending request sequence and server does not have to move anymore. So, the total cost that an optimal algorithm pays is 4 by moving from 6 to 3 following it with moving back to 4 and 6. The figure 4 shows the movement of server for serving type A sequence in optimal case.

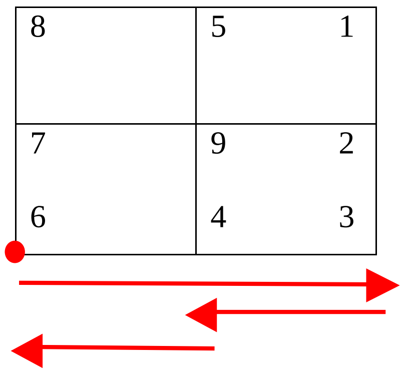

Fig4. For type A, server moves to position 3, 4

And back to 6 with a cost of 4.

For type B, moving the server from 6 to 8 is ideal for initial request to 1 as it automatically serves request to 5 and moreover subsequent 2 requests to 8 and 7. For next request to 2, server moves from 8 to 7 as the further request is to 7 and 6. For request to 4 server moves from 7 to 6 as the further requests are to 3 and 6. Then from the above lemma, we know that staying at position 6 is optimal for ending request sequence and server does not have to move anymore. So, the total cost that optimal algorithm pays is 4 by moving from 6 to 8 following it with moving back to 7 and 6. The figure 5 shows the movement of server for serving type B sequence in optimal case.

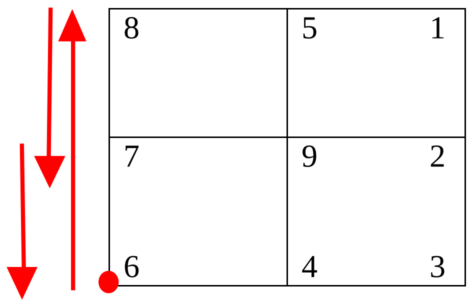

Fig5. For type B, server moves to position 8, 7

And back to 6 with a cost of 4.

It can be realized from the optimal approach that first decision of moving server for request to 1 to position 8 or 3 is a critical respectively for both types of sequence. This means that online algorithm must guess the types of the sequence before serving it. We can reduce this problem to binary string guessing problem such that the type of sequence is encoded as a sequence of binary bits and online algorithm must guess the type of sequence before it is revealed. Consider '1' is encoded as type A and '0' as type B. If online algorithm makes a right guess, then it takes a first right decision and then for the following requests, it just follows a greedy approach and pays same cost of 4 for both A and B as optimal approach. If algorithm makes a wrong guess then surely, it will pay some more cost. Suppose, for type A, online algorithm guesses it as type B. For initial request to 1, online server moves from position 6 to position 8 with a cost of 2. For next request to 2 it follows greedy approach and moves to 7 with a cost of 1. For next request to 3 it moves to position 6 with a cost of 1. It automatically serves next request 4. For request to 5, it moves at position 4 with a cost of 1 and automatically serves 4 and 6. For request to 7 it comes back to position 6 with a cost of 1 and automatically serves rest requests {8, 6, 3, 6, 8, 6, 3, 6, and 8}. Total cost is at least 6, if online server does not make any other wrong decisions. Similarly, if sequence type is wrongly guessed A instead of B then, server moves to 3 for request to 1 with a cost of 2. For next request to 5 it follows greedy approach and moves to 4 with a cost of 1. For next request to 8 it moves to position 6 with a cost of 1. It automatically serves next request 7. For request to 2, it moves at position 7 with a cost of 1 and automatically serves 7 and 6. For request to 4 it comes back to position 6 with a cost of 1 and automatically serves rest requests {3, 6, 8, 6, 3, 6, 8, 6 and 3}. In this case also, online algorithm pays a cost of at least 6 if it takes no further wrong decisions.

For a right guess, the cost is 4 and for a wrong guess, the cost is at least 6. With an advice of 1 bit which can indicate the most frequent type of sequence, online algorithm can make at least half of the guesses correctly. The question arises, how much an algorithm can improve by guessing half of the phases. The competitiveness is analyzed against optimal cost.

Cost of online algorithm with half guesses made correctly = m/2 * 4 + m/2 * 6 = 5m

Optimal Cost for m number of phases = 4m

Competitive Ratio = $\frac{Cost(Alg)}{Cost(Opt)} = \frac{5m}{4m} = 1.25$

Algorithm is 1.25 competitive when it guesses at least half of the sequence types correctly. But in order to be as good as optimal algorithm i.e. to be 1 competitive, it needs to guess more than half of the sequence correctly which needs an advice of a linear function of 'm' (*O*(m)) (from lemma 2). We can conclude a following theorem from the above results.

Theorem 5

For 1-server orthogonal CNN problem, an advice of 1 bit is sufficient to achieve a competitive ratio of 1.25, but in order to be as good as an optimal offline algorithm, still an advice of linear function of n (O(n)) is required.

III. Advice size is less than *O*(n).

We have seen that the lower bound on advice size is tight and an advice of linear function of n of an input request of length n is required to achieve an optimal solution. In this section, we are going to analyze the improvement in the competitive ratio of an online algorithm in a case of 1 server CNN problem when advice size is less than n, say n/c where c is a constant.

Consider an optimal algorithm 'O' processing a 2-dimensional grid R of size (g*g) with 1 server serving the requests of an input sequence of length n appearing in the different locations of the grid. Position of the optimal server at any time can be encoded using (log g, log g) bits of advice each specifying its x and y positions in the grid. We can say that with *O*(log g) bits of advice, position of the optimal server at a particular time can be known. If an algorithm gets an advice of size X (X < n), then it can get some information about the optimal algorithm's decisions while serving the requests. In total, O(X/log g) positions of optimal server can be known. Online algorithm can read this advice and follow the optimal algorithm and brings the server to the exact X/log g positions in the gird. For the rest of the sequences, it follows the greedy approach. Basically, this helps online algorithm to divide the input sequence n into X/log g subsequences of length $\frac{n}{X/\log g}$ $= \frac{n*\log g}{X}$. From theorem 1, we know that the competitive ratio of any algorithm that tends to find the minimum path for every request is at most the length of the sequence, therefore, the competitive ratio in this case will be at most $\frac{n*\log g}{X}$.

If we have linear number of advice bits as X =n/c, c being a constant, then the competitive ratio will be at most $\frac{n*\log g}{n/c} = O(\log g)$. We can see that with linear number of advice bits less than n, a competitive ratio of function of the size of the grid can be achieved. This ratio can be definitely improved when we consider the benefit of having the optimal positions of the server at start of every subsequence. We have decided to work on this later as our future work.

Conclusion

In this paper, we studied the advice complexities of CNN algorithm and its variants. We got some interesting results for the size of advice required in achieving an optimal solution for the CNN problem. For multi-server CNN problem, an advice of size *O*(n*log k) is sufficient to achieve an optimal solution whereas with *O*(n) bits of advice, an optimal solution for 1-server CNN problem can be achieved. The bound on the advice size is tight which we proved using binary string guessing problem which shows that in order to be better than 1.25 competitive for a simple 1 server orthogonal problem of small 2x2 grid, an advice of linear size is required. Moreover, we proved that with a linear number of advice bits, less than the length of input sequence, we can attain the competitive ratio to be at most *O*(log g). This ratio can be definitely improved by considering the benefit of having the optimal positions of the server at start of every subsequence. We leave this work, as our future work.

Bibliography

1. Borodin, Allan, and Ran El-Yaniv. Online computation and competitive analysis. cambridge university press, 2005.
2. S. Dobrev, R. Královiˇc, D. Pardubská, Measuring the problem-relevant information in input, RAIRO Theor. Inform. Appl. 43(3) (2009) 585–613.
3. Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, On the advice complexity of the k-server problem, In Journal of Computer and System Sciences, Volume 86, 2017, Pages 159-170, ISSN 0022-0000, https://doi.org/10.1016/j.jcss.2017.01.001.
4. Nikhil Bansal, Niv Buchbinder, Aleksander M , adry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. J. ACM, 62(5):40:1–40:49, 2015. Preliminary version in FOCS'11. doi:10.1145/2783434.
5. Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. J. ACM, 42(5):971–983, 1995. Preliminary version in STOC'94. doi:10.1145/210118.210128.
6. Elias Koutsoupias and David Scot Taylor. The CNN problem and other k-server variants. Theor. Comput. Sci., 324(2-3):347-359, 2004.
7. Iwama, Kazuo, and Kouki Yonezawa. "The orthogonal CNN problem." Information processing letters 90, no. 3 (2004): 115-120.
8. J. Hromkoviˇc, R. Královiˇc, R. Královiˇc, Information complexity of online problems, in: Proc. MFCS 2010, in: Lect. Notes Comput. Sci., vol.6281, Springer-Verlag, 2010, pp.24–36.
9.]Hans-Joachim B¨ockenhauer, Dennis Komm, Rastislav Kr´aloviˇc, Richard Kr´aloviˇc, and Tobias M¨omke. On the advice complexity of online problems. In ISAAC, volume 5878 of LNCS, pages 331–340, 2009. doi:10.1007/978-3-642-10631-6_35.
10. Gupta, Sushmita, Shahin Kamali, and Alejandro López-Ortiz. "On the Advice Complexity of the k-server Problem Under Sparse Metrics." Theory of Computing Systems 59.3 (2016): 476-499.
11. Boyar, Joan, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. "Online bin packing with

advice." Algorithmica 74, no. 1 (2016): 507-527.

12. Ren‵‵e Sitters, Leen Stougie, and Willem de Paepe. A competitive algorithm for the general 2-server problem. In ICALP '03: 29th International Colloquium on Automata, Languages and Programming, Malaga, Spain, pages 624{636, 2003.
13. Augustine J., Gravin N. (2010) On the Continuous CNN Problem. In: Cheong O., Chwa KY., Park K. (eds) Algorithms and Computation. ISAAC 2010. Lecture Notes in Computer Science, vol 6507. Springer, Berlin, Heidelberg.
14. Böckenhauer, Hans-Joachim, et al. "The string guessing problem as a method to prove lower bounds on the advice complexity." Theoretical Computer Science 554 (2014): 95-108.
15. Class notes.