Randomized Distributed Online Algorithms Against Adaptive Offline Adversaries[☆]

Joan Boyar[a], Faith Ellen[b], Kim S. Larsen[a,*]

[a] *University of Southern Denmark, Department of Mathematics and Computer Science, Campusvej 55, DK-5230 Odense M, Denmark.*
[b] *University of Toronto, Department of Computer Science, 10 King's College Road, Toronto, Ontario M5S 3G4, Canada.*

Abstract

In the sequential setting, a decades-old fundamental result in online algorithms states that if there is a c-competitive randomized online algorithm against an adaptive, offline adversary, then there is a c-competitive deterministic algorithm. The adaptive, offline adversary is the strongest adversary among the ones usually considered, so the result states that if one has to be competitive against such a strong adversary, then randomization does not help. This implies that researchers do not consider randomization against an adaptive, offline adversary. We prove that in a distributed setting, this result does not necessarily hold, so randomization against an adaptive, offline adversary becomes interesting again.

Keywords: online algorithms, randomized algorithms, adaptive offline adversary, distributed systems
2010 MSC: 00-01, 99-00

1. Introduction

The following fundamental result about sequential online algorithms was proved by Ben-David, Borodin, Karp, Tardos, and Wigderson in 1994 [4].

Theorem 1.1. *If there is a c-competitive randomized online algorithm against an adaptive, offline adversary, then there is a c-competitive deterministic algorithm.*

[☆]The first and third authors were supported in part by the Independent Research Fund Denmark, Natural Sciences, grant DFF-7014-00041. The second author was supported in part by the Natural Science and Engineering Research Council of Canada, Discovery Grant RGPIN-2015-05080.

[*]Corresponding author

URL: `imada.sdu.dk/~joan/`; `joan@imada.sdu.dk` (Joan Boyar), `www.cs.toronto.edu/~faith/`; `faith@cs.toronto.edu` (Faith Ellen), `imada.sdu.dk/~kslarsen/`; `kslarsen@imada.sdu.dk` (Kim S. Larsen)

As a consequence, developing randomized algorithms and analyzing them with respect to an adaptive, offline adversary has been considered uninteresting, since one could design a deterministic algorithm instead.

In a survey [2] of the competitive analysis of distributed algorithms, Aspnes observed that distributed online algorithms face uncertainty about the scheduler, in addition to the lack of knowledge of future requests. He compared algorithms for the Repeated Collect problem in a shared memory model to Opt on the same schedule, and established a competitive ratio for the randomized Follow-the-Bodies algorithm [3] against an adaptive, offline adversary. In the Repeated Collect problem, each process owns a register which only it can *write* to. Each process can also repeatedly perform *collect*, which, for each of these registers, returns a value it stores at some point between the beginning and end of the *collect*. A simple algorithm for *collect* is to simply read the registers of the other processes. Follow-the-Bodies is a more complicated algorithm for *collect*, which allows a process to take advantage of information obtained by other processes performing *collect* concurrently.

Aspnes [2] left as an open question whether there is a deterministic algorithm that performs as well as the randomized Follow-the-Bodies algorithm does for the Repeated Collect problem against an adaptive, offline adversary. As explained above, this is a question one would not ask in the sequential setting. In this paper, we show that the question of Aspnes is meaningful in a distributed setting: We exhibit an example where a randomized distributed online algorithm achieves a competitive ratio against an adaptive, offline adversary that no deterministic algorithm can achieve.

2. Online Algorithms and Competitive Analysis

A problem is called *online* if the input is given one piece at a time and an algorithm solving the problem, which is called an *online algorithm*, must make an irrevocable decision regarding each piece before the next piece is given. The pieces of the input are called *requests*. Online algorithms are faced with uncertainty as a result of lack of knowledge about future requests: irrevocable decisions regarding early requests may be unfortunate when future requests arrive. The quality of the solution for a request sequence is measured by an objective function, which is called the *cost* for minimization problems (or *profit* for maximization problems). In the following, we only consider minimization problems.

The most common technique for analyzing online algorithms is competitive analysis [8], where one computes a worst case ratio over all inputs of the cost of the online algorithm's solution to the cost of the solution produced by a (hypothetical) optimal offline algorithm on the same input. Here, *offline* means that the algorithm has access to all of the pieces of the input before making any decisions.

If Alg(I) denotes the cost of running the algorithm Alg on the input sequence I, then Alg is said to be *c-competitive*, if there exists a constant α such that for all I, $\textsc{Alg}(I) \leq c\,\textsc{Opt}(I) + \alpha$, where $\textsc{Opt}$ denotes the minimum possible cost for the input sequence I. When $\alpha = 0$, then $\textsc{Alg}$ is said to be *strictly c-competitive*. The *(strict) competitive ratio* of an algorithm is the infimum over all c for which it is (strictly) c-competitive.

One can think of $\textsc{Opt}$ as a (possibly nonconstructive) algorithm, which, given any input sequence I, solves the problem with cost $\textsc{Opt}(I)$. Note that $\textsc{Opt}$ can behave completely differently on identical prefixes of two different input sequences.

When analyzing deterministic algorithms, we know from the beginning what the algorithm will do on any prefix of the input sequence, and, thus, which state the algorithm will be in. When switching to the randomized case, this is no longer the case, since the state of an algorithm is partially determined by coin flips. Therefore, it makes sense to distinguish between different ways the adversarial input sequence can be constructed (in advance or incrementally), and also when the adversary must make its decision concerning the request (online or offline).

For the sequential setting, there are three types of adversaries that are commonly used against randomized online algorithms [7, 4], listed here in increasing order of strength.

- Oblivious: The adversary constructs the input sequence in advance and the adversary runs $\textsc{Opt}$.
- Adaptive online: The adversary constructs the input sequence incrementally based on the actions of the online algorithm. The adversary must process each request that it generates online before giving it to the online algorithm.
- Adaptive offline: The input sequence is constructed incrementally based on the actions of the online algorithm. The adversary runs $\textsc{Opt}$ on the generated sequence.

In this paper, we focus on adaptive, offline adversaries. Here, the definition of the competitive ratio is similar to that for the deterministic case, but, instead, using the expected values over the coin flips of the online algorithm, i.e., $\textsc{Alg}$ is c-competitive if for all I constructed incrementally by the adversary, depending on the coin flips of the algorithm, $\mathrm{E}\left[\textsc{Alg}(I) - c\ \textsc{Opt}(I)\right] \leq \alpha$.

When solving minimization problems, one can often get smaller competitive ratios with randomized algorithms against an oblivious adversary than for deterministic algorithms. One of the many examples is for the List Access problem: The algorithm COMB [1] achieves a competitive ratio of 1.6 against oblivious adversaries, but no deterministic algorithm can have a competitive ratio better than $2 - \frac{2}{\ell+1}$, where ℓ is the length of the list [6] (which credits Karp and Raghavan). Theorem 1.1 says that a randomized sequential algorithm against an adaptive, online adversary cannot outperform the best deterministic algorithm. The intuition is that the adaptive, offline adversary is so strong that algorithms do not benefit from randomization.

3. The Model and Problem

We consider the following artificial problem, FindValue, designed with the purpose of establishing that Theorem 1.1 does not necessarily hold in a distributed setting. There are 3 processes, p_0, p_1, and p_2 in a synchronous distributed system. Each process, p_i, has one register, R_i to which only it can write. Other processes can get information from p_i by reading its register. In each round, each process can flip some coins and, based on its state and on the outcomes of its coin tosses, it can do nothing, it can write to its register, or it can read the register of some other process. A process cannot choose to do nothing indefinitely: after a finite number of rounds, it must read and write. If there are simultaneous reads to a register which is being written, we assume the read obtains the old value (this is, in fact, a worst-case assumption for the randomized algorithm we present).

Consider the following problem. From time to time, an adversary gives a number as input to one of the processes and lets it take a step (in which it appends a pair consisting of its process ID and this number to its register). In the next round, the adversary notifies each of the other processes that it has produced a new number (by giving each of them a special notification input), but does not tell them to whom it gave this number. The goal is for each process to write the entire sequence of pairs into its register. We assume that the next input number is given only after the two processes that were notified have finished their writes of the current input number to their registers. Our cost (objective) function is the total number of register reads that are performed.

In Opt, each time the adversary notifies a process that it has produced a new number, each of the two other processes will perform one read, from the register of the process that received the number, and append the new pair to its register. Thus, in Opt, two register reads are performed for each input. This will be used to compute the competitive ratios in all of the cases below.

We also note that all results hold for the strict as well as the non-strict (sometimes referred to as *asymptotic*) competitive ratio.

4. Deterministic Upper Bound

The following deterministic algorithm performs 3 reads per input item. Each process p_i maintains a list of the pairs it has learned about and the total number of notifications it has received from the adversary. When the adversary gives a number as input to process p_i, this process appends a pair consisting of i and this number to its list and writes its list to R_i. When notified that a new number is available, process p_i reads from $R_{(i+1) \bmod 3}$. If there are more pairs in that register than in its own list, p_i appends the extra pair to its list and writes its list to R_i. If the length of the list in $R_{(i+1) \bmod 3}$ is smaller than the number of notifications p_i has received from the adversary, then p_i just read from a process having the same list as itself, and in the following round, p_i reads from register $R_{(i-1) \bmod 3}$, appends the extra pair in that register to its list, and writes its list to R_i.

When process p_k gets a number as input directly from the adversary, process $p_{(k-1) \bmod 3}$ will read this number from R_k in the next round. However, process $p_{(k+1) \bmod 3}$ will read from $R_{(k+2) \bmod 3}$ in that round and then will read from R_k in the following round. Thus, a total of three reads are performed for each input. Thus, the algorithm is $\frac{3}{2}$-competitive.

5. Deterministic Lower Bound

An adversary can force 3 reads per input number. For each process p_i, let R_{f_i} be the first location that p_i will read from when informed that the next new number is available. Note that $f_i \neq i$.

Suppose there exist two processes, p_i and p_j, such that $f_i = f_j = k$. Without loss of generality, suppose that $f_k = i$. Then the adversary gives the new input number to p_j. Whichever of p_i and p_k goes first performs at least 2 reads, for a total of 3 reads.

Otherwise, no two processes read from the same register first. Without loss of generality, suppose that $f_0 = 1$, $f_1 = 2$, and $f_2 = 0$. Consider the minimum number of rounds after receiving a notification input before any one of these processes reads a register. Suppose p_k is a process that reads in this round. If the adversary gives the new input number to $p_{(k-1) \bmod 3}$, then p_k reads from $R_{(k+1) \bmod 3}$ in this round and does not see the new number. Hence it has to perform at least 2 reads. Process $p_{(k+1) \bmod 3}$ also has to perform at least one read, for a total of 3. Thus, no deterministic algorithm is better than $\frac{3}{2}$-competitive.

6. Randomized Upper Bound

Consider the following randomized algorithm for FINDVALUE. Each process p_i maintains a list of the pairs it has learned about and the total number of notifications it has received from the adversary. When the adversary gives a number as input to process p_i, this process appends a pair consisting of i and this number to its list and writes its list to R_i.

In this algorithm, the processes access the registers in disjoint rounds. If process p_h is informed at round t that a new number is available, it performs a read at round $t + 3h$ and flips a fair coin to choose from which register R_k (belonging to one of the other two processes) it will read. If there are more pairs in R_k than in its list, p_h appends the extra pair in R_k to its list and writes its list to R_h in the next round. Otherwise, p_h reads from the third register in round $t + 3h + 1$, appends the extra pair contained there to its list, and writes its list to R_h in round $t + 3h + 2$.

Suppose processes p_f and p_ℓ were the two processes informed at round t, where $f < \ell$. Process p_f has probability $1/2$ of choosing to read from R_i. In this case, it only reads once in this round; otherwise, it reads twice. Thus, its expected number of reads is $3/2$. Process p_ℓ will either read from R_i or R_f. In either case, p_ℓ will discover the new number on its first read. Thus, the total expected number of reads is 5/2. Since Opt reads twice, the competitive ratio is 5/4. Thus, there is a randomized algorithm for FindValue that is $\frac{5}{4}$-competitive against an adaptive, offline adversary.

This analysis holds for any adversary that does not have access to future coin flips. In particular, it holds for any oblivious adversary and, more generally, for any adaptive, offline adversary. Because the rounds are independent of each other, the adaptive, offline adversary cannot take advantage of its knowledge of what actions the processes have taken in previous rounds or what coin flips they have used.

7. Discussion

Combining the results of Sections 5 and 6, we have the following result.

Theorem 7.1. *There exists a problem,* FindValue, *in a distributed setting, where there is a randomized algorithm which is $\frac{5}{4}$-competitive against an adaptive, offline adversary, while the best deterministic algorithm is no better than $\frac{3}{2}$-competitive.*

The important observation is that our randomized algorithm against an adaptive, offline adversary has a competitive ratio strictly smaller than the lower bound on any deterministic algorithm. Thus, Theorem 1.1 does not necessarily hold in a distributed setting.

To understand why, it is useful to consider the proof of this result. We give a simplified overview of the proof as explained in the textbook by Borodin and El-Yaniv [5]. An algorithm can be viewed as a tree, where request nodes and answer nodes alternate along every path, starting with a request node at the root. The nodes of the tree represent possible executions of the algorithm. The edges out of a request node represent the possible next requests that an adversary can give. The edges out of an answer node represent the different choices the algorithm can make in response to the request leading into the node. Each of these edges is labeled with the probability that the choice represented by this edge is taken, so the labels of the edges leaving each answer node sum to 1.

The proof depends heavily on the fact that different request sequences and different choices made by the algorithm in response define different nodes in the tree. At each answer node, the algorithm can determine, for any extension of this request sequence, what Opt's cost is. For each of the different choices it can make in response to the request leading into the node and for any extension of the request sequence, it can also determine what its expected cost is, and, hence, which is the best choice. The proof shows that the competitive ratio of the resulting deterministic algorithm is at most the competitive ratio of the randomized algorithm.

In a distributed setting, there is no such algorithm. Each process has only partial information about the request sequence so far and the actions of the other processes. Specifically, it knows the requests it has been given, the steps it has taken, and information it has learned from reading shared registers. In a tree in which the nodes represent the possible executions, there may be many different nodes which are indistinguishable to a particular process. Since the labels of these nodes may be quite different, the process might not be able to determine the best choice to make when allocated a step by the scheduler. For example, in the randomized algorithm in Section 6, when a process is notified that a new number has been produced, it does not know which of the other processes was given the number and, hence, which register to read.

8. Acknowledgments

We thank anonymous referees for their valuable comments.

References

[1] Albers, S., von Stengel, B., Werchner, R., 1995. A combined BIT and TIMESTAMP algorithm for the list update problem. Information Processing Letters 56, 135–139.

[2] Aspnes, J., 1998. Competitive analysis of distributed algorithms. In: Online Algorithms, The State of the Art. Vol. 1442 of Lecture Notes in Computer Science. Springer, pp. 118–146.

[3] Aspnes, J., Hurwood, W., 1998. Spreading rumors rapidly despite an adversary. Journal of Algorithms 26 (2), 386–411.

[4] Ben-David, S., Borodin, A., Karp, R. M., Tardos, G., Wigderson, A., 1994. On the power of randomization in on-line algorithms. Algorithmica 11 (1), 2–14.

[5] Borodin, A., El-Yaniv, R., 1998. Online Computation and Competitive Analysis. Cambridge University Press.

[6] Irani, S., 1991. Two results on the list update problem. Information Processing Letters 38 (6), 301–306.

[7] Raghavan, P., Snir, M., 1989. Memory versus randomization in on-line algorithms. In: 16th International Colloquium on Automata, Languages and Programming (ICALP). Vol. 372 of Lecture Notes in Computer Science. Springer, pp. 687–703.

[8] Sleator, D. D., Tarjan, R. E., 1985. Amortized efficiency of list update and paging rules. Communications of the ACM 28 (2), 202–208.