Volume 5, Issue 2, February 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Review of Online Algorithms and Competitive Analysis

Shivani Garg, Aparajita Nailwal, Mukul Varshney
Department of CSE, Sharda University
Uttar Pradesh, India

Abstract: During the last two decades, online algorithms have received considerable research interest. In this paper, we review the online algorithms and competitive analysis of online algorithms. We study about deterministic algorithms as well as randomized algorithms and discuss how to analyze the randomized algorithms.

Keywords: competitive analysis, deterministic algorithms, online algorithms, randomized algorithms.

I. INTRODUCTION

Traditional optimization techniques assume complete knowledge of all data of a problem instance. However, in real world it is doubtful to expect all information necessary to define a problem instance to be available beforehand. We need to make decisions before complete information is available to us. This ascertainment has motivated the research on *online optimization*. An algorithm is called *online* if it makes a decision (computes a partial solution) whenever a new piece of data requests an action [6].

In online optimization the input is modeled as a finite *request sequence* r_1, r_2, ... which must be served and which is revealed step by step to an online algorithm. The way in which the request sequence is served, depends on the specific *online paradigm*. The two most common models are the *sequence model* and the *time stamp model* [9].

Let ALG be an online algorithm. In the *sequence model* requests must be served in the same order in which they occur. More precisely, when serving request rj, the online algorithm ALG does not know of requests r_i with $i > j$ (or the total number of requests). When request r_j is presented it must be served by ALG according to the specific rules of the problem. The serving of r_j incurs a cost and the overall goal is to minimize the total service cost. The decision by ALG of how to serve rj is irrevocable. Only after r_j has been served, the next request r_{j+1} becomes known to ALG.

In the *time stamp model* each request has a *arrival* or *release time* at which it becomes available for service. The release time $tj \geq 0$ is a nonnegative real number and specifies the time at which request r_j is released (becomes known to an online algorithm). An online algorithm ALG must determine its behavior at a certain moment t in time as a function of all the requests released up to time t and of the current time t. Again, we are in the situation that an online algorithm ALG is confronted with a finite sequence r_1, r_2, ... of requests which is given in order of non-decreasing release times and the service of each request incurs a cost for ALG. The difference to the sequence model is that the online algorithm is allowed to wait and to revoke decisions and that requests need not be served in the order of their occurrence.

Waiting incurs additional costs, typically depending on the elapsed time. Previously made decisions may, of course, only be revoked as long as they have not been executed.

Some of the example online problems are:

- *Resource management in operating systems:* Paging is a classical online problem where a two-level memory system has to be maintained which consists of a small fast memory and a large slow memory. The main aim is to keep actively referenced pages in fast memory without any knowledge of future requests for the pages.
- *Data structures:* Consider a data structure such as a linear linked list or a tree. This structure has to be maintained dynamically so that a sequence of accesses to elements can be served at low cost. Future access patterns are unknown.
- *Scheduling:* It consists of scheduling a sequence of jobs on a set of machines in order to optimize a given objective function. The jobs arrive one by one and no prior knowledge about the future requests is provided. They have to be scheduled immediately.
- *Networks:* Many online problems in this area arise in the context of data transmission. The problem can be, for instance, to dynamically maintain a set of open connections between network nodes without knowing which connections are needed in the future.

The quality of online algorithms is usually evaluated using *competitive analysis*. The idea of competitiveness is to compare the output generated by an online algorithm to the output produced by an *optimal offline algorithm*.

II. ANALYSIS OF ONLINE ALGORITHMS

Online problems had been studied already explicitly or implicitly during the nineteen-seventies and nineteen-eighties.

However, broad systematic investigation only started when Sleator and Tarjan [4] suggested comparing an online algorithm to an *optimal offline algorithm*, thus laying the foundations of *competitive analysis*.

An online algorithm ALG is called c-*competitive* if the objective function value of the solution produced by ALG on any input sequence is at most c times that of an optimal offline algorithm on the same input. Here, the "optimal offline algorithm" has complete knowledge about the whole input sequence.

Observe that in the above definition there is no restriction on the computational resources of an online algorithm. The only scarce resource in competitive analysis is information as quoted by Krumke [6]. Competitive analysis of online algorithms can be imagined as a game between an online player and a malicious offline *adversary*. The online player uses an online algorithm to process an input which is generated by the adversary. If the adversary knows the (deterministic) strategy of the online player, he can construct a request sequence which maximizes the ratio between the player's cost and the optimal offline cost. For randomized algorithms we have to define what kind of information about the online player is available to the adversary.

III. SOME DEFINITIONS

We study the performance of online algorithms in the general framework of request-answer games. In this game an online algorithm has to answer a sequence of requests trying to minimize its cost (as determined by the sequence of requests and answers). The algorithm is online, in the sense that it answers each request before seeing the following requests, and without knowing how long the sequence is. In some games, not all answer sequences are allowed.

A request-answer game [10] consists of a request set R, a finite answer set A, and the cost functions $f_n : R^n \times A^n \rightarrow R \cup \{\infty\}$ for n = 0,1,….. Let f denote the union, over all nonnegative integers n, of the functions f_n. Let us fix such a game.

A deterministic online algorithm G is a sequence of functions $g_i : R^i \rightarrow A$ for i = 1,2, ….. For any sequence of requests $\underline{r} = (r_1, \ldots, r_n)$ we define $G(\underline{r}) = (a_1, \ldots, a_n) \in A^n$ with $a_i = g_i(r_1, \ldots, r_i)$ for i = 1, …. , n. The cost of G on $\underline{r}$ is $c_G(\underline{r}) = f_n(\underline{r}; G(\underline{r}))$. We will compare this to the optimal cost for the same sequence of requests: $c(\underline{r}) = \min\{f_n(\underline{r}, \underline{a}) | a \in A^n\}$ [10].

A randomized online algorithm G is a probability distribution over deterministic online algorithms G_x (x may be thought of as the coin tosses of the algorithm G). For any request sequence $\underline{r}$ the answer sequence $G(\underline{r})$ and the cost $c_G(\underline{r})$ are random variables. We call a randomized online algorithm-competitive if for any $\underline{r}$ we have $E_x(c_{Gx}(\underline{r})) \leq (c(\underline{r}))$.

IV. RANDOMIZED ALGORITHM

An algorithm is called ***randomized*** if its behavior is determined not only by its input but also by the values produced by a ***random-number generator.*** A randomized algorithm flips coins during its execution to determine what to do next. When considering a randomized algorithm, we usually care about its **expected worst-case** performance, which is the average amount of time it takes on the worst input of a given size. This average is computed over all the possible outcomes of the coin flips during the execution of the algorithm.

While studying randomized algorithms, the same issues are taken into consideration as for deterministic algorithms: how to design a good randomized algorithm, and how to prove that it works within given time or error bounds. The main difference is that it is often easier to design a randomized algorithm but harder to analyze it.

Randomized algorithms can be classified by the way they use their randomness to solve a problem. Some very broad categories are:

- **Avoiding worst-case inputs,** by hiding the details of the algorithm from the adversary. Typically it is assumed that an adversary supplies our input. If the adversary can see what our algorithm is going to do, this information can be used against us. If we introduce randomness, our predictable deterministic algorithm is replaced by what is effectively a random choice of many different deterministic algorithms. Since the adversary doesn't know which algorithm we are using, he can't pick an input that is bad for all of them.
- **Sampling**. Here, we use randomness to find an example or examples of objects that are likely to be typical of the population they are drawn from, either to estimate some average value or because a typical element is useful in our algorithm. Randomization means that the adversary can't direct us to non-representative samples.
- **Hashing**. Hashing is the process of assigning a large object *x* a small name *h*(*x*) by feeding it to a **hash function** *h*. Because the names are small, the Pigeonhole Principle implies that many large objects hash to the same name (a **collision**) [11]. If we have few objects that we actually care about, we can avoid collisions by choosing a hash function that happens to map them to different places. Randomization helps here by keeping the adversary from choosing the objects after seeing what our hash function is.
- **Building random structures**. The **probabilistic method** shows the existence of structures with some desired property (often graphs with interesting properties, but there are other places where it can be used) [11] by showing that a randomly-generated structure in some class has a nonzero probability of having the property we want. If we can beef the probability up to something substantial, we get a randomized algorithm for generating these structures.
- **Symmetry breaking**. In **distributed algorithms** involving multiple processes, progress may be stymied by all the processes trying to do the same thing at the same time. Randomization can break these deadlocks.

V. COMPETITIVE ANALYSIS

In competitive analysis, the output generated by an online algorithm is compared to the output generated by an optimal offline algorithm. An optimal offline algorithm already knows the entire input data and thus can result into an optimal output, whereas such is not the case with an online algorithm. The better an online algorithm approximates the optimal solution, the more competitive the algorithm is said to be.

The competitive ratio [10] of an online algorithm for an optimization problem is simply the approximation ratio achieved by the algorithm, that is, the worst-case ratio between the cost of the solution found by the algorithm and the cost of an optimal solution.

Online algorithms can be either deterministic or randomized. The competitive analysis of deterministic online algorithm is straightforward whereas the randomized case is craftier. The competitive ratio of a randomized online algorithm A is defined with respect to an adversary. An adversary can be defined as a pair (Q, S), where Q is the requesting component, responsible for creating the request sequence, and S is the servicing component, which is an algorithm responsible for answering all requests created by Q. The nature of these components will determine the type of adversary- oblivious, adaptive-online or adaptive-offline adversary.

- **Oblivious Adversary [12]:** The oblivious adversary has to generate a complete request sequence in advance, before any requests are served by the online algorithm. The adversary is charged the cost of the optimum offline algorithm for that sequence.
- **Adaptive Online Adversary [12]:** This adversary may observe the online algorithm and generate the next request based on the algorithm's (randomized) answers to all previous requests. The adversary must serve each request online, i.e., without knowing the random choices made by the online algorithm on the present or any future request.
- **Adaptive Offline Adversary [12]:** This adversary also generates a request sequence adaptively. However, it is charged the optimum offline cost for that sequence.

VI. EXTENSIONS OF COMPETITIVE ANALYSIS

Competitive analysis is a sort of worst-case analysis. It has (rightly) been criticized as being overly pessimistic as often the adversary is simply too powerful and allows only insignificant competitiveness results. To conquer this unsatisfactory situation, various extensions and alternatives to pure competitive analysis have been analyzed in the literature [6].

In *comparative analysis* the class of algorithms where the offline algorithm comparative is chosen from is restricted. This concept has been introduced in the context analysis of the paging problem.

Another approach to strengthen the position of an online algorithm is the concept of *resource augmentation* (see e.g. [1,2,3,4]). Here, the online algorithm is given more resources (e.g., more or faster machines in scheduling) to serve requests than the offline adversary.

The *diffuse adversary* model introduced in [5], deals with the situation where the input is chosen by an adversary according to some probability distribution. Although the online algorithm is not aware of the distribution itself, it is provided the information that this distribution belongs to a specific class of distributions.

All of the extensions and alternatives to competitive analysis have been proven to be useful for some *specific problem* and powerful enough to obtain significant results. However, competitive analysis could not be replaced by any of these approaches and is still, the standard tool in the theoretical analysis of online algorithms.

VII. CONCLUSION

From this review, we can conclude that online algorithm is a current research area of interest and competitive analysis has been one of the main tools for deriving worst-case bounds on the performance of algorithms. The competitive analysis of randomized algorithms is more complex than the deterministic ones. Also the competitive analysis has been termed to be overly pessimistic. To deal with the pessimism of competitive analysis, many modifications and alternatives for analyzing online algorithms have been proposed. Hence, competitive analysis can be an important tool for the analysis and optimization of online problems some of which have been cited in the paper.

REFERENCES

[1] C. Phillips, C. Stein, E. Torng, and J. Wein, *Optimal time-critical scheduling via resource augmentation*, *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, 1997, pp. 140.149.

[2] K. Pruhs and B. Kalyanasundaram, *Speed is as powerful as clairvoyance*, *Proceedings of the 36th Annual IEEE Symposium on the Foundations of Computer Science*, 1995, pp. 214.221.

[3] B. Awerbuch, Y. Bartal, and A. Fiat, *Distributed paging for general networks*, *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1996, pp. 574.583.

[4] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, *Communications of the ACM* ***28*** (1985), no. 2, 202.208.

[5] E. Koutsoupias and C. Papadimitriou, *Beyond competitive analysis*, *Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science*, 1994, pp. 394.400.

[6] Sven O Krumke, Lecture Notes on *Online Optimization, World-Wide Web information at* http://optali.com/wp-content/uploads/downloads/2011/03/online-notes.pdf

[7] Rajeev Motwani and Prabhakar Raghavan, *Randomized Algorithms, ACM Computing Surveys, Vol 8, No. 1*, March 1996.

[9] Narendhar Maaroju, K.V.R Kumar, Dr.Deepak Garg, *Approximation Algorithmsfor NP Problems, World-Wide Web information at http://www.academia.edu/1976243/Approximation_Algorithms_for_NP-problems*

[10] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A.Wigderson, *On the power of randomization in on-line algorithms*, *Algorithmica* ***11*** (1994), 2.14.

[11] James Aspnes, Lecture Notes on *Randomized Algorithm, World-Wide Web information at http://www.cs.yale.edu/homes/aspnes/classes/469/notes-2011.pdf*

[12] Susanne Albers, *Online Algorithms, World-Wide Web information at http://www2.informatik.hu-berlin.de/~albers/papers/inter.pdf*

