Benchmark of Bitrate Adaptation in Video Streaming

JESSICA CHEN, University of Windsor, Canada
HENRY MILNER, Conviva, USA
ION STOICA, University of California, Berkeley, USA
JIBIN ZHAN, Conviva, USA

The HTTP adaptive streaming technique opened the door to cope with the fluctuating network conditions during the streaming process by dynamically adjusting the volume of the future chunks to be downloaded. The bitrate selection in this adjustment inevitably involves the task of predicting the future throughput of a video session, owing to which various heuristic solutions have been explored. The ultimate goal of the present work is to explore the theoretical upper bounds of the QoE that any ABR algorithm can possibly reach, therefore providing an essential step to benchmarking the performance evaluation of ABR algorithms. In our setting, the QoE is defined in terms of a linear combination of the average perceptual quality and the buffering ratio. The optimization problem is proven to be NP-hard when the perceptual quality is defined by chunk size and conditions are given under which the problem becomes polynomially solvable. Enriched by a global lower bound, a pseudo-polynomial time algorithm along the dynamic programming approach is presented. When the minimum buffering is given higher priority over higher perceptual quality, the problem is shown to be also NP-hard, and the above algorithm is simplified and enhanced by a sequence of lower bounds on the completion time of chunk downloading, which, according to our experiment, brings a 36.0% performance improvement in terms of computation time. To handle large amounts of data more efficiently, a polynomial-time algorithm is also introduced to approximate the optimal values when minimum buffering is prioritized. Besides its performance guarantee, this algorithm is shown to reach 99.938% close to the optimal results, while taking only 0.024% of the computation time compared to the exact algorithm in dynamic programming.

CCS Concepts: • **Information systems** → **Multimedia streaming**; • **Theory of computation** → Dynamic programming; • **Networks** → *Network performance modeling;*

Additional Key Words and Phrases: Video delivery, adaptive bitrate algorithm

ACM Reference format:
Jessica Chen, Henry Milner, Ion Stoica, and Jibin Zhan. 2021. Benchmark of Bitrate Adaptation in Video Streaming. *J. Data and Information Quality* 13, 4, Article 22 (August 2021), 24 pages.
https://doi.org/10.1145/3468063

Authors' addresses: J. Chen, School of Computer Science, University of Windsor, 401 Sunset Avenue, Windsor, ON, Canada, N9B 3P4; email: xjchen@uwindsor.ca; H. Milner and J. Zhan, Conviva, 989 E Hillsdale Blvd. #400, Foster City, CA, 94404; emails: {hmilner, jibin}@conviva.com; I. Stoica, Computer Science Division, University of California, Berkeley, 465 Soda Hall, Berkeley, CA, 94720-1776; email: istoica@cs.berkeley.edu.

1936-1955/2021/08-ART22 $15.00
https://doi.org/10.1145/3468063

1 INTRODUCTION

With the rapidly increasing popularity of Internet video, the online delivery of video content now dominates a major fraction of the Internet traffic. The competition of the network resource for rapid data transmission over the Internet leads to typical phenomena like the *freezing screen* or the *blurring display*, which seriously affect the viewer's satisfaction and engagement [7, 16, 20]. How to achieve higher ***quality of experience*** **(QoE)** in the delivery of video content is being extensively studied. One of the essential issues to the study is the adaptation of the content delivery to different network conditions.

The network condition for a video session is affected by many factors like the geographical location of the player, the connectivity type of the video session, the device type, and so forth. It also varies along the time horizon in the sense that the download speed over the Internet does not stay constant throughout an entire viewing session. The HTTP adaptive streaming technique has emerged in this regard to cope with the diversified and dynamically changing network conditions. With this technique, a video is chopped into *chunks*, each associated with several files to choose from. Selecting a lower bitrate for a chunk, for example, results in a smaller chunk size, which could be more suitable when the network condition is poor. Applications of this technique include, for example, the HTTP Live Streaming from Apple and the HTTP Dynamic Streaming from Adobe.

How to dynamically select a suitable bitrate for each chunk is normally carried out by a process running an ***adaptive bitrate*** **(ABR)** algorithm, either on the server or on the client side, according to the network conditions. Whether a selected bitrate is suitable or not depends on the future network condition during which this chunk is downloaded. Consequently, this selection inevitably involves predicting the future throughput of the current video session. Owing to this prediction, various heuristic solutions have been explored in recent years [2, 5, 6, 9, 11, 13, 15, 17, 18, 23, 25–27]. Some of them provide different design logics [6, 11, 26], while some others offer different ways to reach more suitable parameters through customization [2] or more suitable models through learning [17, 18]. See [4] for a summary of various state-of-the-art bitrate adaptation algorithms.

The performance among ABR algorithms is largely impacted by different techniques on throughput prediction. Here we take the viewpoint that the throughput prediction is 100% correct. The ultimate goal is to find the theoretical upper bounds of the QoE that any ABR algorithm can possibly reach, therefore providing an essential step to benchmarking the evaluations of ABR algorithms.

During a session of video play, the throughput rate can be periodically measured. A piecewise function of time obtained from connecting these throughput rates can then be brought into the study on the improvement of ABR algorithms (see, e.g., [2, 12, 24]). In an effort to build benchmark models, we take the throughput trace from a played video session and define the optimal bitrate sequence on it where the bitrates are selected with the full knowledge of the future throughput rates at each decision point. In this sense, we define the *optimality*: an optimal value on each throughput trace defines the upper bound of the QoE obtained from any ABR algorithm applied to this trace. With the factor of the prediction removed, the present work differs from existing ones involving the analysis and experiment based on a given set of throughput traces for the development of online ABR algorithms [1, 2, 8, 17, 18, 22, 24]. Note that our optimization model is defined on throughput traces instead of bandwidth traces because bandwidth information is hard to retrieve from real systems.

A number of metrics have been defined to quantitatively measure the quality of the video display from the perspective of the video viewers. In this article, we consider join time a parameter to the model, and QoE is defined in terms of a linear combination of the average perceptual quality and the buffering ratio.

Our study shows that, in the above setting, the optimization problem is NP-hard when the perceptual quality is defined by chunk size. Limited to **constant bitrate (CBR)** encoding, the problem is NP-hard when the number of bitrates is considered a variable, and polynomially solvable otherwise. Enriched by a global lower bound, we present a pseudo-polynomial-time algorithm along the dynamic programming approach. When the minimum buffering is given higher priority over perceptual quality, in the sense that minimizing buffering is considered more important than any improvement of the perceptual quality, the problem is proven to be also NP-hard, and the above algorithm is simplified and enhanced by a sequence of lower bounds on the completion time of chunk downloading. According to our experiment, this brings a 36.0% performance improvement in terms of computation time. To facilitate the processing of a large amount of data, a more efficient polynomial-time algorithm is introduced to the problem with minimum buffering prioritized. Internally called *gold standard*, this algorithm was implemented in Conviva Inc. in 2015. Since then, it has served within the company as a benchmark for the performance evaluation of the ABR algorithms. In this article, the algorithm is proven to provide an approximation to the optimal value with a guaranteed lower bound. Compared to the exact algorithm in the dynamic programming approach, it gets 99.938% close to the optimal results while taking only 0.024% of the computation time.

The rest of the article is organized as follows: In Section 2, we provide a brief introduction to the preliminary background and related work. The problem definitions are presented in Section 3, after a summary of our working context and the notations used in this article. In Section 4, we present our results on the problem complexity. The general dynamic programming algorithm is given in Section 5, and we present in Section 6 its enhancement for the special setting where minimum buffering is prioritized. The greedy approximation is introduced in Section 7, together with the claim on its performance. Included in Section 8 are the comparisons between the proposed benchmark and two parameterized online ABR algorithms, followed by the performance comparisons among our general dynamic programming approach, its enhancement when minimum buffering is prioritized, and the greedy approximation. The conclusion and some final remarks are found in Section 9.

2 PRELIMINARIES AND RELATED WORK

In this section, we give a brief summary of the QoE measurement proposed in the literature, followed by an introduction to existing work on ABR approaches.

2.1 Quality of Experience

Quality display in terms of viewer experience is measured in various aspects. *Join time*, *buffering ratio*, *rate of buffering events*, *average bitrate*, and so forth are all well-addressed QoE metrics [3, 7]. Mostly defined on a single video session, QoE metrics can also be interpreted on a group of video sessions [1, 21].

The *join time* refers to the period counted from the moment the user clicks on a video for viewing to the moment when the video starts to play. During the join time, the play buffer gets loaded with one or more initial chunks.

The *buffering time* is the total amount of time in a session when the video gets stopped (i.e., frozen) because the play buffer is drained. *Buffering ratio* is the percentage of the buffering time over the total amount of play time or session time. The *rate of buffering events* measures the number of times buffering occurs within a video session.

A video comes with a set of quality levels, each with a targeted bitrate (also called bitrate candidate in the following) averaged across all the chunks in the entire video. The selection of a bitrate from this set of candidates for a specific chunk determines the chunk file associated with

this bitrate to be downloaded. The *average bitrate* for a video session for QoE measurement refers to the average of the bitrates among all the chunks downloaded during this viewing session. It is an indicator of the overall display quality in a video session.

The total *quality switch* refers to the total number of bitrate changes in a session.

2.2 Related Work

The QoE metrics like those listed above are very often conflicting to each other, so the performance of an ABR algorithm has to be a trade-off among them. The following ABR examples illustrate some different handling of this trade-off in regard to the rendering quality:

- In MPC [26], four QoE metrics are considered: (1) average bitrate, (2) buffering time, (3) join time, and (4) quality switch. The overall quality of experience is defined by their *linear combination*, each metric associated with a given coefficient representing the weight of the corresponding metrics that should be taken into account. A higher weight is given to the *total bitrate changes*, for example, when bitrate changes are considered an essential impact factor to the viewer's satisfaction and engagement, and this guides the bitrate adaptation to put in more effort to reduce the changes.
- According to BBA [11], the bitrate is determined by the state of the play buffer via a predefined mapping function. The emphasis on different QoE metrics is reflected in the coefficients of this function.

In addition to making bitrate selection based on different principles about the trade-off of QoE metrics, the ABR algorithms also distance each other by adopting different strategies of predicting the throughput for downloading the future chunks. In some ABR designs, this prediction is more explicit (see, e.g., [6, 13, 26]). In others, it is implicit.

- In MPC [26], the throughput prediction is based on the average throughput rate recorded during the downloading of the previous chunks, with the assumption that network conditions are reasonably stable on short timescales.
- In ELASTIC [6], feedback control theory is adopted to construct a controller to fill up the buffer to a certain length in order to avoid future buffering. The calculation includes the speed of the changes of the buffer length, which, to some extent, represents the predicted future network condition.
- In BBA [11], the bitrate selection is based solely on the amount of data the client currently possesses in the play buffer, not explicitly on the prediction of the throughput rate. The throughput prediction is embedded in the coefficients of its mapping function.
- In HYB [2], both the predicted throughput and the current buffer state are explicitly used to determine the next bitrate. A higher predicted throughput value and larger buffer state are contributors to the selection of a higher bitrate.

The ABR model proposed in MPC [26] is expressed in terms of an objective function and a set of constraints. Although designed for the application in the actual systems, it could also be considered an optimization model without the impact factor of the prediction, provided that (1) the throughput remains a constant and (2) the sliding window used to limit the computation is no smaller than the total number of chunks in the video session.

The idea and some initial work appeared in the literature in recent years on computing an offline optimal ABR solution based on recorded throughput traces for the purpose of building a benchmark for the evaluation of ABR solutions. In [19], Gurobi is adopted to solve two offline optimal ABR problems. One is to maximize the average bitrate with no occurrence of buffering. The other is to minimize the quality switches. The authors pointed out that these are known as

the *multiple-choice nested knapsack problem* and *quadratic multiple-choice nested knapsack problem,* respectively, both NP-hard. The detailed proof of the first one appears in [10]. Mostly about online ABR design, the work of [23] also includes an implementation of an offline algorithm for optimal ABR along the dynamic programming approach.

3 CONTEXT, NOTATIONS, AND PROBLEM DEFINITION

We work in the following context: (1) Each video is associated with a given set of bitrates. (2) The size of each chunk at each bitrate in a session is known. (3) For any chunk, selecting a bigger bitrate implies getting a larger chunk size and receiving higher perceptual quality. (4) All chunks are of an equal *duration*, which is measured by their amount of *play time*. (5) The bitrate selection is always performed upon the completion of each chunk. (6) The play buffer is large enough to keep the download procedure from pausing. (7) The throughput trace of each session is given in terms of a piecewise function of time. Note that the results presented in this work are not affected by how frequently the throughput rates are measured or how the throughput function is defined from the measured rates, provided that, upon the comparison between an optimal offline solution and that from a particular online ABR algorithm, the same throughput functions are applied.

We will be using the following notations:

- $maxbr$ denotes the total number of bitrate candidates.
- $rmin$ and $rmax$ denote the smallest and largest bitrate candidates, respectively.
- $maxch$ denotes the total number of chunks in a session.
- $chdr$ denotes the chunk duration.
- $chsz(r, i)$ denotes the size of chunk i at bitrate r ($1 \le i \le maxch$).
- $q(r, i)$ denotes the perceptual quality of chunk i at bitrate r ($1 \le i \le maxch$).
- jt denotes the join time.
- $bf([r_1, \ldots, r_k])$ denotes the total amount of buffering time when downloading the k chunks with bitrate sequence $[r_1, \ldots, r_k]$, starting from time 0.
- $\mathcal{T}$ denotes the given piecewise throughput function. Each video session for analysis has its own throughput function starting from time 0.
- $\mathcal{E}(t, i, [r_1, \ldots, r_k])$, for $0 \le i \le maxch - 1$ and $k \ge 1$, denotes the completion time of downloading a sequence of chunks $i+1, \ldots, i+k$, from the given starting time t, with given throughput $\mathcal{T}$ (omitted from the notation for simplicity), and a given sequence of bitrates $r_1, \ldots, r_k$. $\mathcal{E}(t, i, (r, k))$ is an abbreviation of $\mathcal{E}(t, i, [r_1, \ldots, r_k])$ when $r_1 = \cdots = r_k = r$.
- $(a)_+$ takes value a when $a \ge 0$ and value 0 when $a < 0$.

Note that $chsz(r, i), q(r, i), bf([r_1, \ldots, r_k]), \mathcal{E}(t, i, [r_1, \ldots, r_k])$ may not be functions. They are written with parenthesis to avoid double subscript.

The QoE of a video session is a linear combination of its average perceptual quality avp and its buffering ratio bfr, expressed by function

$$qoe(avp, bfr) = avp - \alpha * bfr,$$

with a buffering penalty coefficient α to express the desired ratio between the two QoE metrics. When α increases, the QoE gets more sensitive to the buffering time.

avp is the average of the perceptual quality $q(r, i)$ for each chunk i at its selected bitrate r. Although precisely measuring the perceptual quality $q(r, i)$ may not be easy, there are various ways to approximate it. The chunk size $chsz(r, i)$, for example, is a possible choice of the measurement (see, e.g., [10, 19]). Note that a specific measurement of the perceptual quality $q(r, i)$ could bring in additional conditions to the optimization problems. Setting $q(r, i)$ to be $chsz(r, i)$, for example, implies that the optimization problems are considered in a special setting with the added relationship

between $q(r, i)$ and the download completion time. The presence of additional conditions could have an impact on the study of the problems considered. In this article, we present our results for both the setting with the general definition of $q(r, i)$ and the setting where $q(r, i)$ is measured by $chsz(r, i)$: (1) For the problem complexity, we present our results for the special setting where $q(r, i)$ is set to $chsz(r, i)$, and the complexity in the general setting of $q(r, i)$ follows. (2) The algorithms and the related performance guarantee presented in this work are valid for any setting of $q(r, i)$.

Given any coefficient α, join time jt, throughput function $\mathcal{T}$, total number of chunks $maxch$, chunk duration $chdr$, and a set of bitrate candidates, the ***maximum linearly combined QoE (MLQ)*** *problem* is to find a sequence of $maxch$ bitrates to maximize QoE. Let r_i denote the bitrate selected for the ith chunk and b_i the buffering time during the downloading of the ith chunk. The problem is to maximize

$$\frac{1}{maxch}\sum_{i=1}^{maxch} q(r_i, i) - \alpha * \frac{1}{chdr \times maxch} bf([r_1, \ldots, r_{maxch}]),$$

where

$$bf([r_1, \ldots, r_{maxch}]) = \sum_{i=1}^{maxch} b_i$$

$$b_i = \left(\mathcal{E}(0, 0, [r_1, \ldots, r_i]) - jt - (i-1) * chdr - \sum_{j=1}^{i-1} b_j\right)_+ \qquad i = 1, \ldots, maxch.$$

$bf([r_1, \ldots, r_{maxch}])$ can also be expressed by the maximum of the tardiness among each chunk downloading: $bf([r_1, \ldots, r_{maxch}]) = max\{tn_i([r_1, \ldots, r_i]) \mid 1 \le i \le maxch\}$. Here, $tn_i([r_1, \ldots, r_i])$ is the tardiness of the ith chunk downloading:

$$tn_i([r_1, \ldots, r_i]) = (\mathcal{E}(0, 0, [r_1, \ldots, r_i]) - jt - (i-1) * chdr)_+.$$

Join time is a parameter to our model, and there is no restriction on its value. It does not need to guarantee the existence of a buffering-free bitrate sequence. If the join time is set small and the download of the first chunk could not get finished before that time, the difference between the download completion time of the first chunk and the join time is counted as buffering time.

From the viewpoint of multi-objective optimization, the MLQ optimization falls in the line of the *weighted sum* approach to reach the nondominated or Pareto optimal solutions.

When minimizing buffering time is given higher priority over improving perceptual quality, we keep the buffering time at its minimum. Clearly, this problem is single-objective. The minimum buffering time, denoted by *minbuf*, is obtained by adopting the smallest bitrate candidate *rmin* for all chunks. The problem, denoted by MB-MLQ, then amounts to maximizing the average perceptual quality while keeping the buffering time to this *minbuf*—that is, to maximize

$$\frac{1}{maxch}\sum_{i=1}^{maxch} q(r_i, i)$$

s.t.

$$\mathcal{E}(0, 0, [r_1, \ldots, r_i]) \le jt + (i-1) * chdr + minbuf \qquad i = 1, \ldots, maxch.$$

4 COMPLEXITY

Theorem 4.1. *If the perceptual quality is defined by chunk size, the MLQ problem is NP-hard.*

Proof. We prove this claim by reducing the subset sum problem to the MLQ decision problem. Given $S = \{a_1, \ldots, a_n\}$ and K (for simplicity, $n > 1$), the subset sum problem is to determine the existence of a subset $S' \subseteq S$ so that $\sum_{a_i \in S'} a_i = K$. The MLQ instance is constructed as follows: (1) $\alpha = chdr * n$. (2) The number of chunks in a session is n. (3) The set of bitrate candidates is $\{1, 2\}$. (4) $chsz(1, i) = chdr$, $chsz(2, i) = chdr + chdr * n * a_i$ $(1 \le i \le n)$. (5) $jt = chdr$. (6) The throughput function

$$\mathcal{T}(t) = \begin{cases} 1 + n * K & 0 \le t < chdr \\ 1 & t \ge chdr. \end{cases}$$

The MLQ *decision problem* is to determine, for any given Q, whether there exists a bitrate sequence $[br_1, \ldots, br_n]$ with $QoE \ge Q$. Let $Q = chdr + chdr * K$.

For any $S' \subseteq S$ where $\sum_{a_i \in S'} a_i = K$, let $br_i = 1$ if $a_i \notin S'$ and $br_i = 2$ if $a_i \in S'$. Then, the total buffering time $b = (chdr * n + chdr * n * \sum_{a_i \in S'} a_i - chdr * (1 + n * K) - chdr * (n - 1))_+ = chdr * n * (\sum_{a_i \in S'} a_i - K)_+ = 0$. $QoE = (\sum_{i=1}^{n} chsz(br_i, i))/n = (chdr * n + chdr * n * \sum_{a_i \in S'} a_i)/n = Q$. So, we have $QoE \ge Q$.

On the other hand, for any given bitrate sequence $[br_1, \ldots, br_n]$ with $QoE \ge Q$, let $S' = \{a_i \mid br_i = 2\}$. We have that $QoE = chdr + chdr * \sum_{a_i \in S'} a_i - chdr * n * (\sum_{a_i \in S'} a_i - K)_+$. In the following, we show that it is not possible to have $\sum_{a_i \in S'} a_i < K$ or $\sum_{a_i \in S'} a_i > K$.

(1) If $\sum_{a_i \in S'} a_i < K$, then $QoE = chdr + chdr * \sum_{a_i \in S'} a_i \ge Q = chdr + chdr * K$, which gives $\sum_{a_i \in S'} a_i \ge K$ (contradiction).
(2) If $\sum_{a_i \in S'} a_i > K$, then $QoE = chdr + chdr * \sum_{a_i \in S'} a_i - chdr * n * (\sum_{a_i \in S'} a_i - K) \ge Q = chdr + chdr * K$, which implies $(n - 1) * (K - \sum_{a_i \in S'} a_i) \ge 0$. This gives $\sum_{a_i \in S'} a_i \le K$ (contradiction).

Thus, $\sum_{a_i \in S'} a_i = K$. □

Theorem 4.2. *If the perceptual quality is defined by chunk size, the MB-MLQ problem is NP-hard.*

The MB-MLQ *decision problem* is to determine, for any given Q, whether there exists a bitrate sequence $[br_1, \ldots, br_n]$ so that the total buffering time is *minbuf* and $QoE \ge Q$. The proof is analogous to that of Theorem 4.1. Note that in the proof of Theorem 4.1, $minbuf = 0$ according to the construction of the instance. Note also that item (2) is not needed for the proof of MB-MLQ because any solution to the MB-MLQ problem implies that there is no buffering. Hence, $\sum_{a_i \in S'} a_i - K \le 0$.

The MLQ problem is a generalization of the *MLQ problem with perceptual quality defined by chunk size*. So the MLQ problem is also NP-hard. Analogously, the MB-MLQ problem is NP-hard.

The ***subset sum* (SS)** is a well-known NP-hard problem widely adopted to analyze the complexity of various problems and to exemplify the reduction technique in complexity theory. Similar to the *knapsack problem,* which has come along with variants like the *bounded knapsack problem*, the *unbounded knapsack problem*, the *knapsack problem with fixed number of items,* and so forth, the SS problem is also followed by several interesting variants. Here, we consider the ***subset sum with fixed subset size* (SSF)** problem and the ***unbounded subset set with fixed subset size* (USSF)** problem.

Given a set S of natural numbers, and natural numbers k and K, the SSF problem is to determine whether there exists a subset S' of S with cardinality k so that the total sum of the numbers in S' is K. This is a variant of the SS problem in the sense that the cardinality of S' must be the given number k. The USSF problem is different from the SSF problem in that S' is a multiset, and there is no bound on how many times a same number could occur in S'.

The SSF problem is NP-hard: Following the reduction from the ***3-dimension matching* (3DM)** problem to the SS problem (see, e.g., [14]), we obtain straightforwardly the reduction from the 3DM

problem to the SSF problem. The USSF problem can also be proven to be NP-hard, the conclusion of which will be used in this article to prove Theorem 4.4. We outline the proof of the NP-hardness of USSF in the following, based on a reduction from the SSF problem.

Theorem 4.3. *USSF problem is NP-hard.*

Proof outline: Consider any SSF instance with set $S = \{a_1, \ldots, a_n\}$, cardinality k of the subset, and the total sum K. Let B be an arbitrary number greater than $a_1, \ldots, a_n$. Let

$$b_i = (2^{n+1} + 2^i) * (n+1)^2 * B \qquad i = 1, \ldots, n$$
$$c_i = (2^{n+1} + 2^i) * (n+1)^2 * B + a_i * (n+1) + 1 \qquad i = 1, \ldots, n$$
$$K' = \left(n * 2^{n+1} + \sum_{i=1}^{n} 2^i\right) * (n+1)^2 * B + K * (n+1) + k.$$

Define an instance of USSF with set $\{b_1, \ldots, b_n, c_1, \ldots, c_n\}$, cardinality n of the subset, and the total sum K'. K' takes a sum of three terms. The first term of K' is used to make sure that any solution L to the USSF instance must be a *set* (i.e., without repeating elements), and for each i ($1 \le i \le n$), either b_i or c_i, with an exclusive *or*, must be in the solution. The second term of K' is used to guarantee that $\sum_{c_i \in L} a_i = K$. The third term of K' is used to make sure that $|\{i \mid c_i \in L\}| = k$. It can be proved that each solution to the SSF instance corresponds to a solution to the USSF instance and vice versa.

Theorem 4.4. *Let perceptual quality be defined by chunk size. With CBR, if the number of bitrate candidates is not fixed, the MLQ problem is NP-hard.*

Proof. We prove the NP-hardness by reducing the USSF problem to the MLQ decision problem. Given any USSF instance with set $S = \{a_1, \ldots, a_n\}$, k, and K, we construct an MLQ instance as follows: (1) $\alpha = chdr * k$. (2) The set of bitrate candidates is $\{a_1, \ldots, a_n\}$. (3) The number of chunks in a session is k. (4) $chsz(a_i, j) = chdr + chdr * k * a_i$ ($1 \le i \le n$, $1 \le j \le k$). (5) $jt = chdr$. (6) The throughput function

$$\mathcal{T}(t) = \begin{cases} 1 + k * K & 0 \le t < chdr \\ 1 & t \ge chdr. \end{cases}$$

The MLQ *decision problem* is to determine, for any given Q, whether there exists a bitrate sequence $[br_1, \ldots, br_n]$ with $QoE \ge Q$. Let $Q = chdr + chdr * K$.

For any multiset $S' = \{b_1, \ldots, b_k\}$ from S where $\sum_{i=1}^{k} b_i = K$, consider the bitrate sequence $[b_1, \ldots, b_k]$. The total buffering time is $chdr * k * (\sum_{i=1}^{k} b_i - K)_+ = 0$. $QoE = (\sum_{i=1}^{k} chsz(b_i, i))/k = Q$. So, we have $QoE \ge Q$.

On the other hand, for any given bitrate sequence $[br_1, \ldots, br_k]$ with $QoE \ge Q$, let multiset $S' = \{br_1, \ldots, br_k\}$. We have that $QoE = chdr + chdr * \sum_{i=1}^{k} br_i - chdr * k * (\sum_{i=1}^{k} br_i - K)_+$. In the following, we show that it is not possible to have $\sum_{i=1}^{k} br_i < K$ or $\sum_{i=1}^{k} br_i > K$.

(1) If $\sum_{i=1}^{k} br_i < K$, then $QoE \ge Q$ implies $\sum_{i=1}^{k} br_i \ge K$ (contradiction).
(2) If $\sum_{i=1}^{k} br_i > K$, then $QoE \ge Q$ implies $\sum_{i=1}^{k} br_i \le K$ (contradiction).

So we have $\sum_{i=1}^{k} br_i = K$. □

Theorem 4.5. *Let perceptual quality be defined by chunk size. With CBR, if the number of bitrate candidates is not fixed, the MB-MLQ problem is NP-hard.*

The proof is analogous to that of Theorem 4.4. Note that in the proof of Theorem 4.4, $minbuf = 0$ according to the construction of the instance. Note also that item (2) is not needed for the proof of

MB-MLQ because any solution to the MB-MLQ problem implies that there is no buffering. Hence, $\sum_{i=1}^{k} br_i \leq K$.

Theorem 4.6. *With CBR, if the number of bitrate candidates is fixed, the MLQ problem and the MB-MLQ problem can be solved in time* $O(maxch^{maxbr})$.

Proof. In the context of CBR, $chsz(r, i) = chsz(r, j)$ (for any r, i, j), and we work in the setting where $r \leq r'$ implies $chsz(r, i) \leq chsz(r', i)$. Let $[r_1, \ldots, r_n]$ be an optimal bitrate sequence, and $[r'_1, \ldots, r'_n]$ its permutation in nondecreasing order. Then $[chsz(r'_1, 1), \ldots, chsz(r'_n, n)]$ is a nondecreasing permutation of $[chsz(r_1, 1), \ldots, chsz(r_n, n)]$. This implies that using bitrate sequence $[r'_1, \ldots, r'_n]$ instead of $[r_1, \ldots, r_n]$ will not increase the buffering time. Furthermore, it does not change the perceptual quality. Therefore, for either the MLQ or MB-MLQ problem, $[r'_1, \ldots, r'_n]$ is also an optimal sequence. This means that in the effort of searching for an optimal solution, we could enumerate all bitrate sequences in nondecreasing order, which amounts to determining how many times each bitrate is selected. For each bitrate, there are at most $maxch + 1$ different choices. So, there are at most $(maxch + 1)^{maxbr}$ distinct sequences in nondecreasing order. We can examine each sequence to determine which one yields the maximum QoE. □

5 DYNAMIC PROGRAMMING

In regard to problems with high complexity, dynamic programming, when applicable, very often offers a promising approach to the solutions. Given that the MLQ problem is proven to be NP-hard in our setting, we show in this section that the problem could be resolved in pseudo-polynomial time along the dynamic programming approach. To improve the efficiency, a global lower bound is embedded into the algorithm. Note that this algorithm (called DP in the following) is based on the quantization of the session length and the buffering time. Therefore, rigorously speaking, it is an approximate algorithm.

The bitrates in each video session are selected iteratively from the first chunk to the last one. The decision on the bitrate selection for the ith chunk depends solely on the information obtained from the computation on the $(i-1)$th chunk. We maintain two matrices P and N at any time during the execution to keep the information of the previous chunk and of the next chunk. Suppose that we are currently in the iteration for the $(i + 1)$th chunk. If $P[t][b]$ is not null, it represents the existence of a bitrate sequence for the initial i chunks ending with download completion time t and total buffering time b, and among the sums of the perceptual quality values of those sequences with t and b, $P[t][b]$ keeps the largest one.

The soundness of this algorithm is based on the following property: Let $s = [r_1, \ldots, r_i]$ and $s' = [r'_1, \ldots, r'_i]$ be two bitrate sequences for the initial i chunks that lead to the same download completion time t and same total buffering time b. $\sum_{j=1}^{i} q(r_j, j) \leq \sum_{j=1}^{i} q(r'_j, j)$. Then for any bitrate candidate r, $[r_1, \ldots, r_i, r]$ and $[r'_1, \ldots, r'_i, r]$ will also lead to the same download completion time and the same total buffering time. Therefore, between s and s', we only need to keep s' because s will not lead to any better QoE than what s' can lead to.

The algorithm is enriched by a global lower bound for efficiency. A partial bitrate sequence can be removed if it will not lead to any optimal sequence. Given the perceptual quality and buffering time for a partial bitrate sequence of the initial k chunks ($1 \leq k \leq maxch - 1$), we can derive a possible QoE from it for the entire video session assuming that *rmax* is used for all the rest of the $maxch - k$ chunks and no buffering occurs during the downloading of these $maxch - k$ chunks. The obtained QoE is the maximum that this partial sequence can possibly lead to. If this QoE value cannot compete with the lower bound, the partial bitrate sequence is removed from consideration. Choosing *rmin* for all chunks provides a possible lower bound for this purpose.

The program runs in time $O(maxch * n_1 * n_2 * maxbr)$, where n_1 is the quantized maximum session length, and n_2 is quantized maximum buffering time.

For CBR, we only need to consider bitrate sequences in nondecreasing order. To do so, we record in matrix P and N the largest r among all those bitrates already selected in the previous i chunks. Then only bitrates no smaller than r will be considered for the $(i+1)$th chunk.

6 MINIMUM BUFFERING CONSTRAINTS

When minimum buffering is given higher priority, the QoE can be obtained from the above DP algorithm by setting a large α. In this section, we present a more efficient algorithm (called DP0 in the following) designed specifically for MB-MLQ. This efficiency enhancement is based on the following property in the MB-MLQ problem.

Proposition 6.1. *If* $\mathcal{E}(0,0,[r_1,\ldots,r_k]) = \mathcal{E}(0,0,[r'_1,\ldots,r'_k])$, $bf([r_1,\ldots,r_k]) \le minbuf$, $bf([r'_1,\ldots,r'_k]) \le minbuf$, *then for any* $r''_{k+1},\ldots,r''_{maxch}$, $bf([r_1,\ldots,r_k,r''_{k+1},\ldots,r''_{maxch}]) = minbuf$ *implies* $bf([r'_1,\ldots,r'_k,r''_{k+1},\ldots,r''_{maxch}]) = minbuf$.

Proof. $\mathcal{E}(0,0,[r_1,\ldots,r_k]) = \mathcal{E}(0,0,[r'_1,\ldots,r'_k])$ implies that $tn_i([r_1,\ldots,r_k,r''_{k+1},\ldots,r''_i]) = tn_i([r'_1,\ldots,r'_k,r''_{k+1},\ldots,r''_i])$ for $k+1 \le i \le maxch$. Then we have $maxtn = maxtn'$, where

$$maxtn = max\{tn_i([r_1,\ldots,r_k,r''_{k+1},\ldots,r''_i]) \mid k+1 \le i \le maxch\}$$
$$maxtn' = max\{tn_i([r'_1,\ldots,r'_k,r''_{k+1},\ldots,r''_i]) \mid k+1 \le i \le maxch\}.$$

Since $bf([r_1,\ldots,r_k,r''_{k+1},\ldots,r''_{maxch}]) = minbuf$, we have $maxtn \le minbuf$. So $maxtn' \le minbuf$. This, together with $bf([r'_1,\ldots,r'_k]) \le minbuf$, gives $bf([r'_1,\ldots,r'_k,r''_{k+1},\ldots,r''_{maxch}]) \le minbuf$ and the conclusion follows. □

Let s and s' be two bitrate sequences for the initial k chunks where s has a higher quality than s'. According to Proposition 6.1, if s and s' have the same download completion time and both have buffering time no greater than *minbuf*, then we can discard the information of s', *no matter what buffering time it has*, because for any bitrate sequence s'', if s' concatenated with s'' is a feasible MB-MLQ sequence, then s concatenated with s'' is also a feasible MB-MLQ sequence.

Since there is no need to keep the information about the buffering time, we derive DP0 from DP by replacing the matrix $P[t][b]$ (and $N[t][b]$) with an array $P[t]$ (and $N[t]$). If $P[t]$ is not null, it represents the existence of a bitrate sequence for the initial k chunks ending with download completion time t and total buffering time no greater than *minbuf*, and among the sums of the quality values of these sequences, $P[t]$ keeps the largest one.

To make sure that P[t] records information of only those sequences whose total buffering time is no greater than *minbuf*, we introduce a sequence of lower bounds on the download completion time. These bounds provide conditions stronger than guaranteeing that a (partial) bitrate sequence has a total buffering time no greater than *minbuf*: it also guarantees that it can lead to at least one feasible solution. For a bitrate sequence $[r_1,\ldots,r_k]$ with download completion time t, we put condition on time t to make sure that the total buffering time of the entire session does not exceed *minbuf* if chunks $i+1,\ldots,maxch$ are all downloaded at bitrate *rmin*. Note that this condition implies that the total buffering time of $[r_1,\ldots,r_k]$ is no greater than *minbuf*. This bound on t is the largest time t' satisfying

$$\mathcal{E}(t',i,(rmin,j)) \le jt + chdr \times (i+j-1) + minbuf \qquad \forall j.\ 1 \le j \le maxch - i.$$

These bounds are expressed in array *latest*, one per chunk index. For chunk i, $latest[i]$ is the latest time required to complete the download of the initial i chunks in order to guarantee minimum buffering if all the rest of the chunks are downloaded with *rmin*. If the download of chunk i cannot be completed by this latest time, any bitrate sequence selected for the rest of the chunks will lead to more buffering than *minbuf*, which means the sequence can be discarded.

ALGORITHM 1: Latest[i] for minimum buffering constraints

for $i = maxch$ *to* 1 **do**
  $latest[i] = jt + chdr \times (i-1) + minbuf$;
  if $i < maxch$ **then**
    $latest[i] = min\{latest[i], prev(latest[i+1], rmin)\}$;

The values of $latest[i]$ for $1 \le i \le maxch$ can be computed once for all before the bitrate selection starts. The computation is carried out in a backward manner. For chunk $maxch$, $latest[maxch]$ is the time when chunk $maxch$ is about to play with $minbuf$ of delay, i.e., $jt + chdr \times (maxch - 1) + minbuf$. For chunk i where $i < maxch$, $latest[i]$ is the smaller one between (1) the time when chunk i is about to play with $minbuf$ of delay, i.e., $jt + chdr \times (i-1) + minbuf$, and (2) $prev(latest[i+1], rmin)$. Here, for any t, we use $prev(t, r)$ to denote the time to start downloading a chunk at bitrate r so that the completion time is t.

The pseudocode of *latest* is given in Algorithm 1. The entire DP0 algorithm runs in time $O(maxch * n * maxbr)$, where n is the quantized maximum session length.

In the setting of CBR, we can make use of the nondecreasing order to achieve better performance in terms of computation time. Let $[r_1, \ldots, r_i]$ be a bitrate sequence where r is the largest in it. Since any bitrate selected in the future will only be r or higher, we could extend the *latest[i]* into a two-dimensional array $latest[r][i]$. For bitrate candidate r and chunk i, $latest[r][i]$ is the latest time required to complete the downloading of the initial i chunks in order to guarantee minimum buffering if all the rest of the chunks are downloaded with r. The calculation of $latest[r][i]$ follows that of $latest[i]$ straightforwardly.

7 GREEDY APPROXIMATION

To handle a large amount of data, more efficient algorithms are in demand, even though this means some possible compromise to the optimality. Given that the optimization problems are set up for benchmarking purposes, a heuristic algorithm is acceptable only if it could provide answers always within a known distance from the optimal ones. We present here an efficient algorithm for the MB-MLQ problem. It runs in time $O(maxch * maxbr)$ with guaranteed lower bound.

Following the greedy strategy, this algorithm consists of two essential steps: (1) Use play time constraints to mark, for each chunk, the latest download completion time $latest[i]$ that guarantees the minimum buffering. (2) Select, for each chunk, the highest bitrate so that its file can be completely downloaded before the latest download completion time for this chunk.

The pseudocode is given in Algorithm 2. Let t_i $(1 \le i \le maxch)$ be the download completion time of chunk i $(t_0 = 0)$. The bitrate selected for the ith chunk is

$$max\{r_k \mid 1 \le k \le maxbr,\ \mathcal{E}(t_{i-1}, i-1, [r_k]) \le latest[i]\}.$$

Condition $\mathcal{E}(t_{i-1}, i-1, [r_k]) \le latest[i]$ is used to make sure that selecting bitrate r_k will result in a download completion time that can lead to at least one complete bitrate sequence without causing more buffering than *minbuf*.

Now we show that the result from this greedy algorithm approximates the optimal value with a guaranteed lower bound. Given the selected bitrate r for chunk i, define weight

$$w(r, i) = \frac{chsz(r, i)}{q(r, i)}.$$

The lower bound of the greedy algorithm is defined with the maximum and the minimum of the weights $w(r, i)$:

ALGORITHM 2: Greedy algorithm

```
calculate latest;
total = 0;
t = 0;
for i from 1 to maxch do
    for j from maxbr to 1 do
        t' = E(t, i − 1, [r_j]);
        if t' ≤ latest[i] then
            total = total + r_j;
            t = t';
            break;
return total/maxch;
```

- $minw = min\{w(r_i, j) \mid 1 \le i \le maxbr,\ 1 \le j \le maxch\}$
- $maxw = max\{w(r_i, j) \mid 1 \le i \le maxbr,\ 1 \le j \le maxch\}$

Let $r_1, \ldots, r_{maxbr}$ be the given bitrate candidates in nondecreasing order. Then $q(r_1, i), \ldots, q(r_{maxch}, i)$ is also in nondecreasing order for $1 \le i \le maxch$. The lower bound of the performance is related to the largest difference between two adjacent quality values:

$$maxdiff = max\left\{q(r_{k+1}, i) - \frac{minw}{maxw} \times q(r_k, i) \mid 1 \le k \le maxbr - 1,\ 1 \le i \le maxch\right\}.$$

Let $[gr_1, \ldots, gr_{maxch}]$ $(1 \le i \le maxch)$ be the bitrate sequence selected for each chunk according to the greedy strategy. Let $[or_1, \ldots, or_{maxch}]$ $(1 \le i \le maxch)$ be the optimal bitrate sequence. The performance guarantee of the algorithm is expressed in the following statement.

Theorem 7.1.

$$\frac{1}{maxch}\sum_{i=1}^{maxch} q(gr_i, i) \ge \frac{minw}{maxw}\frac{1}{maxch}\sum_{i=1}^{maxch} q(or_i, i) - \frac{maxdiff}{maxch}.$$

Proof. We prove by induction on the length of the sequence of the chunks that for all j $(1 \le j \le maxch)$,

$$\sum_{i=1}^{j} q(gr_i, i) \ge \frac{minw}{maxw} \times \sum_{i=1}^{j} q(or_i, i) - maxdiff.$$

(1) Base step $(j = 1)$:

gr_1 and or_1 both get applied to the same starting time 0, and they both guarantee the minimum buffering time of the session. According to the way gr_1 is selected, we have $gr_1 \ge or_1$, which implies $q(gr_1, 1) \ge q(or_1, 1)$. So we have

$$q(gr_1, 1) \ge \frac{minw}{maxw} \times q(or_1, 1) - maxdiff.$$

(2) Induction step $(j = k + 1)$:

Assume that we have the hypothesis

$$\sum_{i=1}^{k} q(gr_i, i) \ge \frac{minw}{maxw} \times \sum_{i=1}^{k} q(or_i, i) - maxdiff.$$

We prove the inequation for $k+1$ in two situations:

(2.1) gr_{k+1} is the maximal bitrate candidate.

In this case, $gr_{k+1} \geq or_{k+1}$, which implies $q(gr_{k+1}, k+1) \geq q(or_{k+1}, k+1)$. So, according to the hypothesis, the inequation for $k+1$ holds.

(2.2) gr_{k+1} is not the maximal bitrate candidate.

Let gr'_{k+1} be the smallest bitrate candidate that is greater than gr_{k+1}: $gr'_{k+1} = min\ \{r | gr_{k+1} < r\}$. Since gr_{k+1} is selected according to the greedy strategy, we know that $\mathcal{E}(0, 0, [gr_1, \ldots, gr_k, gr'_{k+1}])$ cannot lead to any complete bitrate sequence with buffering time no greater than *minbuf*. $[or_1, \ldots, or_k, or_{k+1}]$, on the other hand, can do so. Therefore,

$$\mathcal{E}(0, 0, [gr_1, \ldots, gr_k, gr'_{k+1}]) \geq \mathcal{E}(0, 0, [or_1, \ldots, or_k, or_{k+1}]),$$

which implies

$$\sum_{i=1}^{k+1} q(gr_i, i) * w(gr_i, i) + \delta \geq \sum_{i=1}^{k+1} q(or_i, i) * w(or_i, i),$$

where $\delta = (q(gr'_{k+1}, k+1) * w(gr'_{k+1}, k+1) - q(gr_{k+1}, k+1) * w(gr_{k+1}, k+1))$. So we have

$$\sum_{i=1}^{k+1} q(gr_i, i) \geq \frac{minw}{maxw} \sum_{i=1}^{k+1} q(or_i, i) - maxdiff. \qquad \square$$

8 EXPERIMENT

Included in this section are the experimental results related to the proposed methods. The experiment is based on a proprietary dataset from Conviva Inc. The backlogged data consists of 201,904 throughput rate recordings for 8,408 records of sessions, together with a set of seven bitrate candidates. The bitrate values fall in the range of (150, 2500). The chunk duration is 6s. The throughput functions obtained from the recordings are piecewise constant functions. The throughput traces from different sessions vary widely in terms of their lengths. Those traces from long sessions are trimmed to 300s in order to keep the memory consumption and computation time reasonable during the execution of DP and DP0. The perceptual quality is approximated by bitrate and measured by kbps. The average perceptual quality is also called average bitrate in this section. The buffering time is measured by milliseconds.

In regard to the results of the computation time, the programs are written in C and executed on a server machine running GNU/Linux with x86_64 instruction set on 24-core Genuine Intel(R) CPU at 2.50GHz.

This section is organized as follows. In Section 8.1, the optimal values discussed in this work are compared with those from two configurable online ABR algorithms. We first consider the optimal values obtained from DP, then those obtained from DP0. Section 8.2 is dedicated to the characteristics of the proposed DP algorithm, which is demonstrated through the variation of the join time as well as the variation of the α values. Experiments related to DP0 are presented in Section 8.3, where the effectiveness of the simplification of the data structure and the introduction of the lower bound to the download completion time in DP0 is given via the performance comparison between DP and DP0. In Section 8.4, results regarding the greedy algorithm are presented: the performance comparison between DP0 and its greedy approximation shows how much we gain when the latter is put into practice.

8.1 Benchmark at Work

Sided by two ABR algorithms in the literature, we demonstrate in this section the proposed benchmark at work. The two ABR algorithms we have sampled are both simple for our presentation and

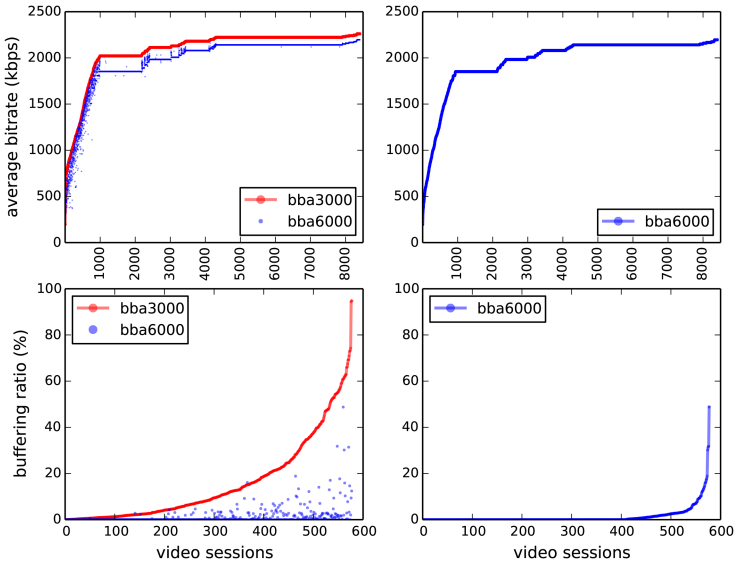

Fig. 1. Average bitrate and buffering ratio from BBA with different configurations.

configurable for illustrating the comparisons with parameters both in the ABR solutions and in our proposed methods.

We have taken two examples of α values, 5,000 and 20,000. To obtain QoE from DP, the program is executed twice with two different values of α. To obtain QoE from a sample ABR, the ABR program is executed only once: with the average bitrate and buffering ratio obtained from the execution, the QoE is calculated with different α values. The join time is the same in both DP and the sample ABR algorithms.

The first ABR example is the BBA algorithm proposed in [11]. According to this algorithm, a piecewise function is defined to map the states of the play buffer to the bitrate candidates. This mapping function is introduced with parameters. The first parameter, called a *reservoir*, represents the minimum required buffer length to avoid future buffering. If this reservoir is set high, the ABR algorithm would prefer that the bitrate stay at its minimum. The second parameter, called *slope*, represents the speed to increase the buffer length to the desired range. The algorithm will be more aggressive when the slope is set high.

In our experiment, the BBA algorithm is exemplified with two pairs of configurations: (1) reservoir = 3,000ms, slope = (maxbr-1)/3,000; (2) reservoir = 6,000ms, slope = (maxbr-1)/6,000. They are labeled by bba3000 and bba6000, respectively. bba3000 is more aggressive in choosing higher bitrates and taking higher risk of buffering. This is shown in Figure 1 where the bottom two figures are drawn with only those 578 sessions that have at least one nonzero value in bba3000 and bba6000.

The comparison between BBA and DP is shown in Figure 2 with α = 5,000 and α = 20,000. When α = 5,000, the averages of the QoE values from DP, bba3000, and bba6000 are 2,249.23, 2,025.69, and 1,965.90, respectively. When α = 20,000, the averages of the QoE values from DP, bba3000, and bba6000 are 2,210.23, 1,863.93, and 1,953.05, respectively. Between the figures with bba3000 (the first and the third in the row) and those with bba6000 (the second and the fourth in the row), we see how the gaps from the QoE of the ABR solution to the QoE of the DP solution change when the former moves from a more aggressive to a more conservative configuration. Between the figures

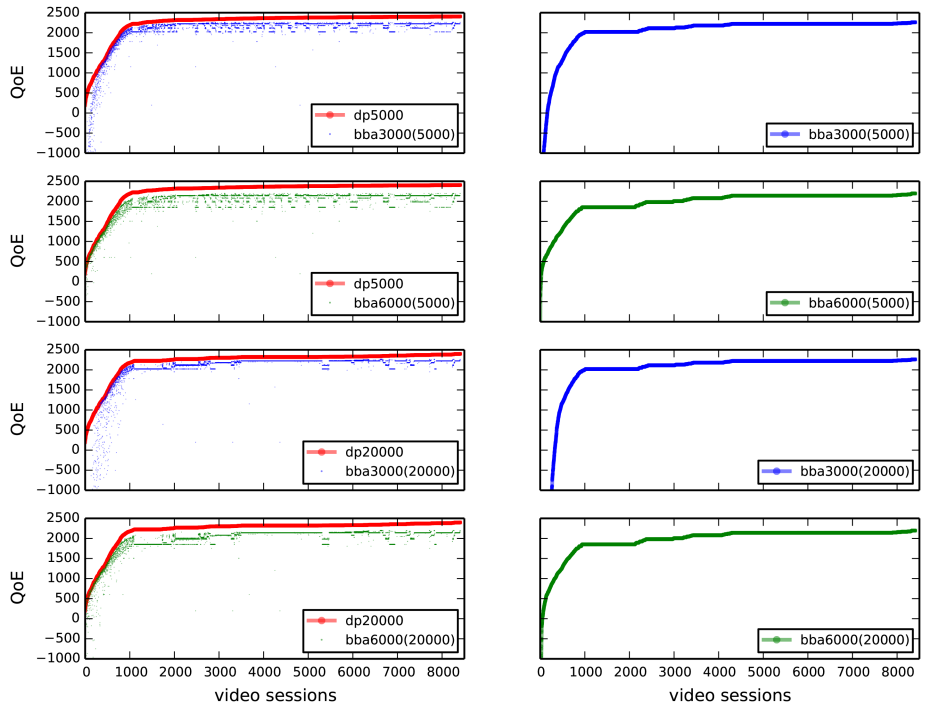

Fig. 2. QoE comparison between DP and BBA.

with $\alpha = 5{,}000$ (the first and the second in the row) and those with $\alpha = 20{,}000$ (the third and the fourth in the row), we see the change of the gaps from the QoE of the same ABR program and the same configuration to the QoE of the DP, due to the change of our QoE goal expressed via the value of α.

The second sample ABR algorithm, called HYB [2], considers both the predicted throughput and the buffer state to determine the next bitrate. Let p be the predicted throughput based on past samples, and *len* the current buffer length. For each chunk i, HYB picks the largest bitrate r satisfying condition $chsz(r, i) < \beta * len * p$. Like the reservoir and slope in BBA, β is a parameter in this formula. A higher β value will lead to more aggressive bitrate selection.

In our experiment, HYB is exemplified with two configurations: $\beta = 0.3$ and $\beta = 0.8$. The predicted throughput is set to the average of five samples of past throughput rates in the same session. Figure 3 shows the average bitrate and buffering ratio of hyb03 and hyb08, where the bottom two figures are drawn with only those 86 sessions that have at least one nonzero value in hyb03 and hyb08. The buffering issue is very well handled in both hyb03 and hyb08: hyb03 has only 10 nonzero values and hyb08 has only 86.

The comparison between HYB and DP is shown in Figure 4 with $\alpha = 5{,}000$ and $\alpha = 20{,}000$. When $\alpha = 5{,}000$, the averages of the QoE values from DP, hyb03, and hyb08 are 2,249.23, 2,039.08, and 2,084.15, respectively. When $\alpha = 20{,}000$, the averages of the QoE values from DP, hyb03, and hyb08 are 2,210.23, 2,038.47, and 2,078.17, respectively. Since the buffering happens very rarely in HYB, there is not much difference between the QoE values with $\alpha = 5{,}000$ and those with $\alpha = 20{,}000$ in hyb03 and hyb08, respectively.

In addition to DP, the online ABR algorithms could also be compared with DP0. With this comparison, there is no α involved, and the average bitrate and buffering ratio are compared separately.

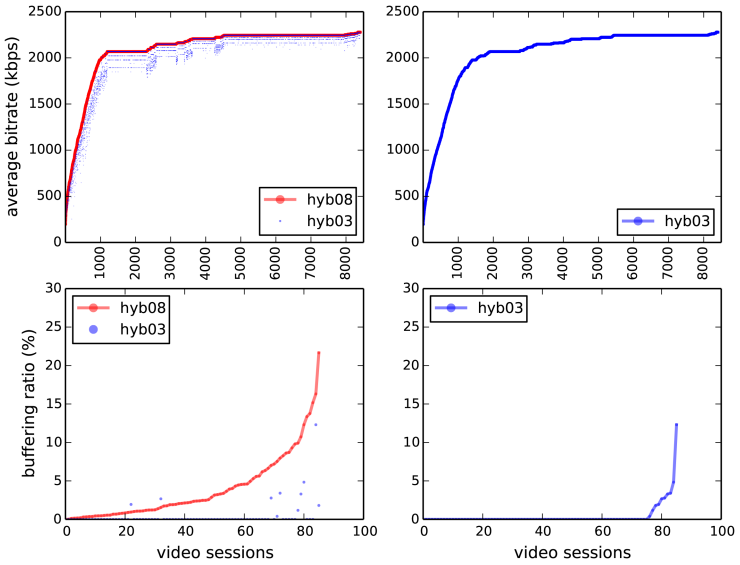

Fig. 3. Average bitrate and buffering ratio from HYB with different configurations.

For the comparison of the buffering ratio between DP0 and the two sample ABR algorithms, we have set the join time in a way so that *minbuf* is zero. Therefore, the comparison could be found in Figure 1 and Figure 3, where the buffering ratios from DP0 are all zero and are not plotted. bba3000 and bba6000 have an average buffering ratio valued at 1.0784% and 0.0857%, respectively. hyb03 and hyb08 have an average buffering ratio 0.0041% and 0.0399%, respectively.

The comparison of average bitrate among DP0, bba3000, and bba6000 is shown in Figure 5, with the bitrate averaged across all chunks valued at 2,205.97, 2,079.61 and 1,970.19 respectively. The comparison of average bitrate among DP0, hyb03, and hyb08 is shown in Figure 6, with the bitrate averaged across all chunks valued at 2,205.97, 2,039.29 and 2,086.14, respectively.

8.2 Characteristics of DP

The DP algorithm is parameterized by the join time and the α value. Here, we show the variations of the benchmark data according to the given parameter values.

When join time increases, the potential of buffering is reduced. So the QoE value is likely to get increased. Figure 7 shows the QoE variations with join time 50ms, 500ms, and 1,000ms. The average QoE values obtained from these join time values are 2,208.42, 2,241.39, and 2,261.75 respectively. The increase of the QoE value comes from either the increase of the average bitrate or the decrease of the buffering ratio, or both. The average bitrate for different join times is given in Figure 8. The average bitrate from DP across all sessions results in values of 2,259.16, 2,263.12, and 2,270.91, respectively, for the three join time values. The buffering ratio for different join times is presented in Figure 9, with the average of 0.507%, 0.217%, and 0.091%,and the number of zero values 15, 4,125, and 6,090, respectively.

The increase of the α value leads to the selection of lower bitrate in order to reduce the buffering time. The QoE values very often tend to decrease. We have set α = 5,000, 10,000, and 20,000 to demonstrate this trend. Figure 10 shows the QoE with these three α values. On average, the QoE are valued at 2,249.21, 2,228.50, and 2,210.22, respectively. Figure 11 shows the average bitrate with these three α values. Their average bitrates across all sessions are valued at 2,277.77, 2,258.64, and 2,231.31, respectively. Figure 12 shows the buffering ratio with the three α values, the average being 0.570%, 0.301%, and 0.105%, respectively.

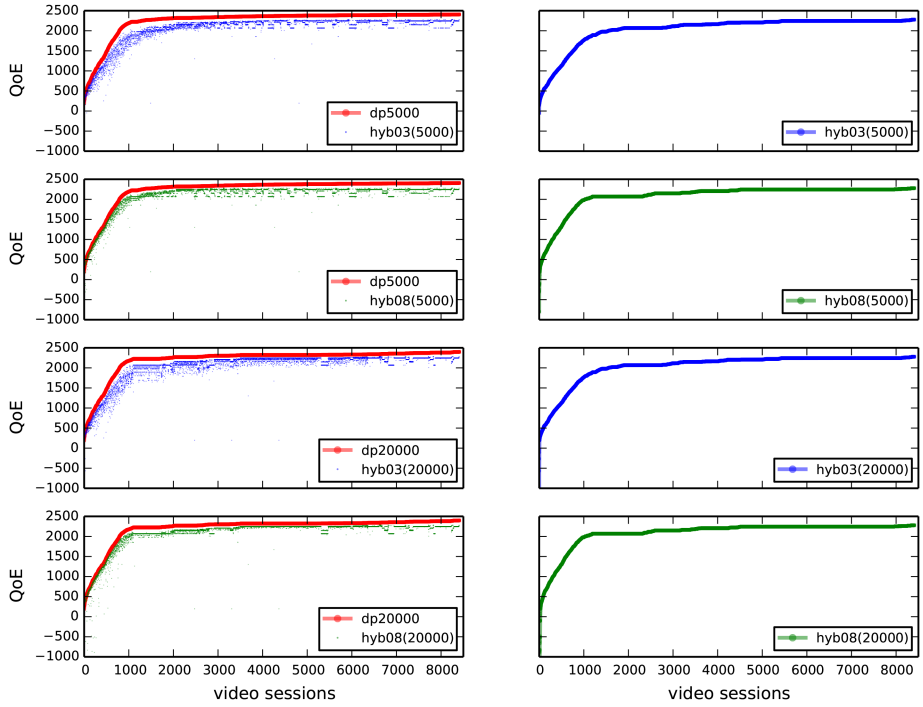

Fig. 4. QoE comparison between DP and HYB.

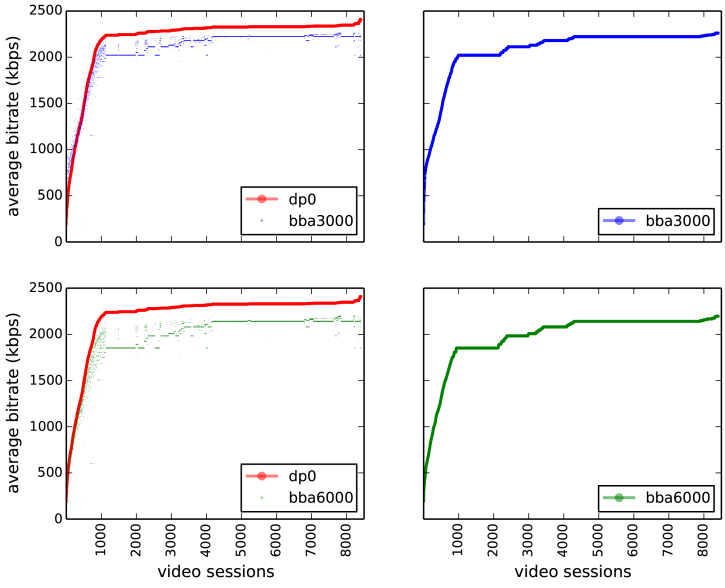

Fig. 5. Comparison of average bitrate between DP0 and BBA.

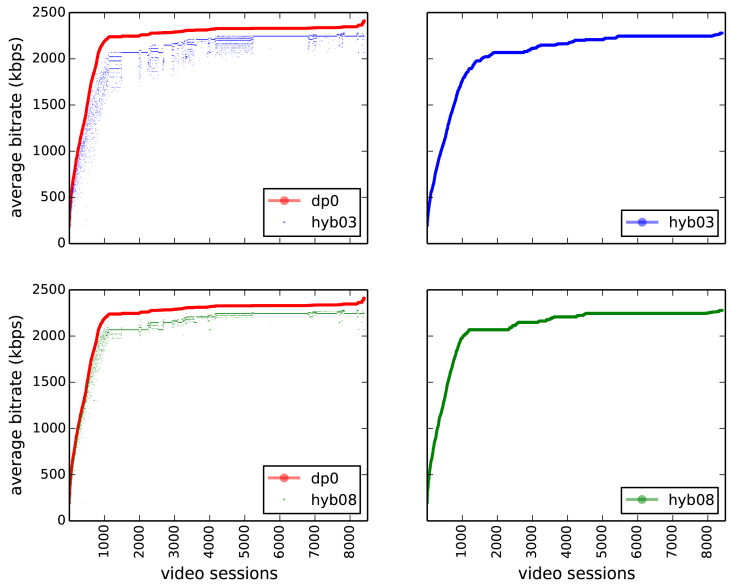

Fig. 6. Comparison of average bitrate between DP0 and HYB.

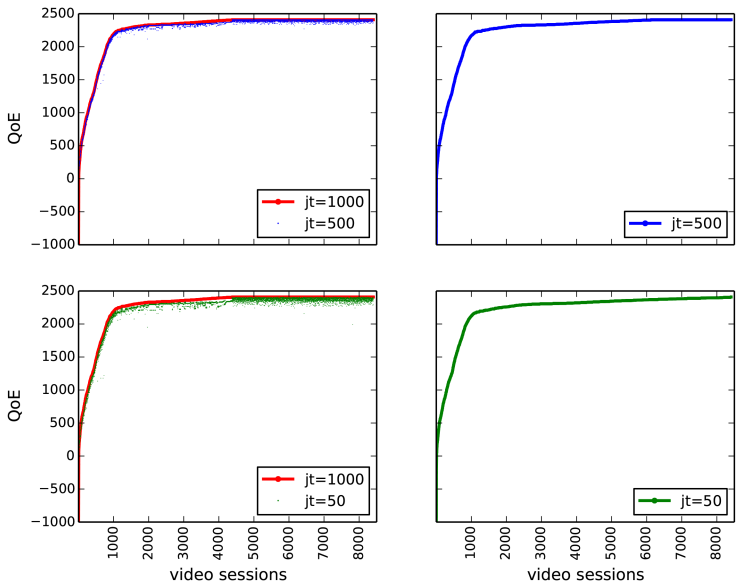

Fig. 7. QoE from DP with different join times.

8.3 Efficiency of DP0

When the minimum buffering is prioritized, the optimal value can be obtained from running either DP (with a large α) or DP0. The latter is less demanding on memory occupancy and comes with added constraints on the download completion time. Figure 13 shows the comparison of the computation time between DP and DP0 from our experiment. α is set to 20,000. To obtain a matrix

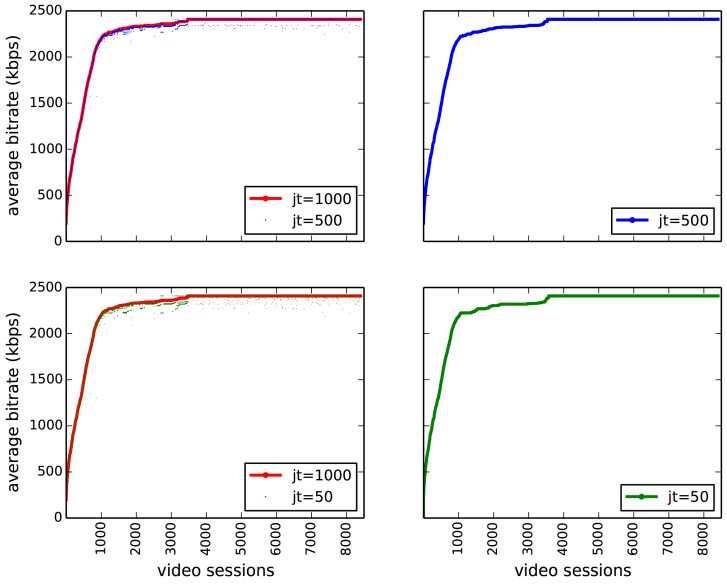

Fig. 8. Average bitrate from DP with different join times.

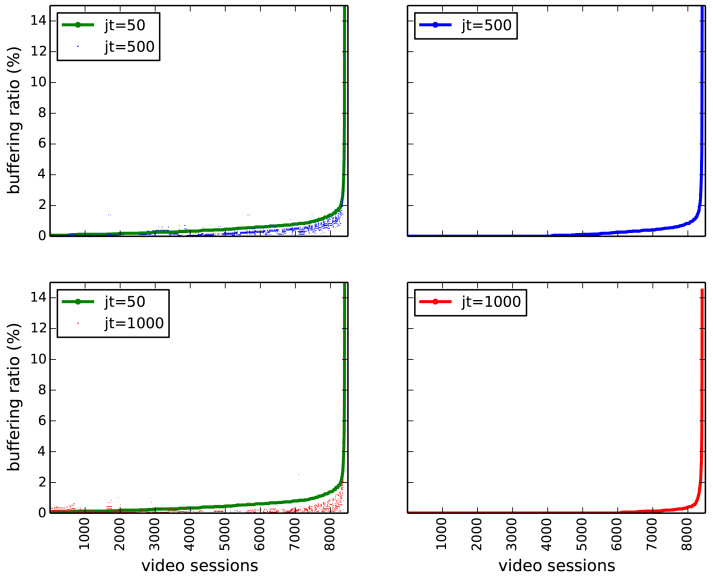

Fig. 9. Buffering ratio from DP with different join times.

or an array indexed by the download completion time, the two algorithms used granularity of 1ms to quantize the session time. To obtain a matrix indexed by the buffering time, DP used time granularity of 100ms. On average, the execution of DP0 takes 46.61ms, giving 36.0% improvement compared to the average of 72.78ms from DP. Note that this percentage should be interpreted together with the session length and the granularity of time.

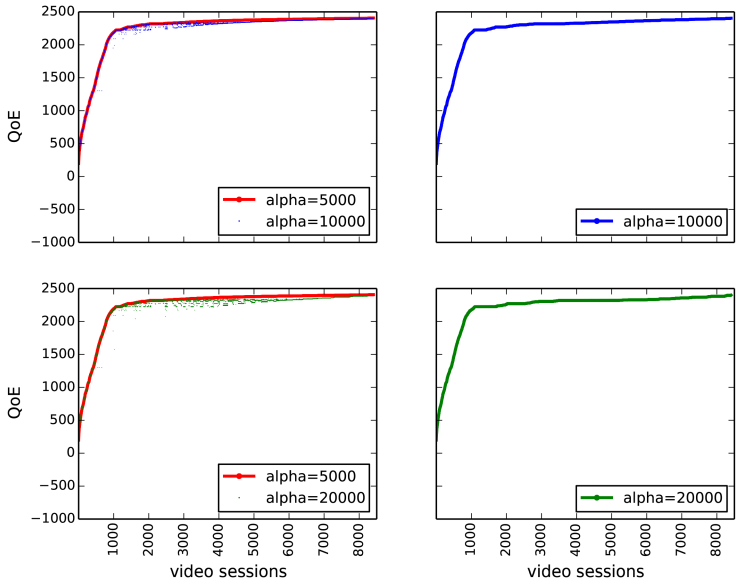

Fig. 10. QoE from DP with different α values.

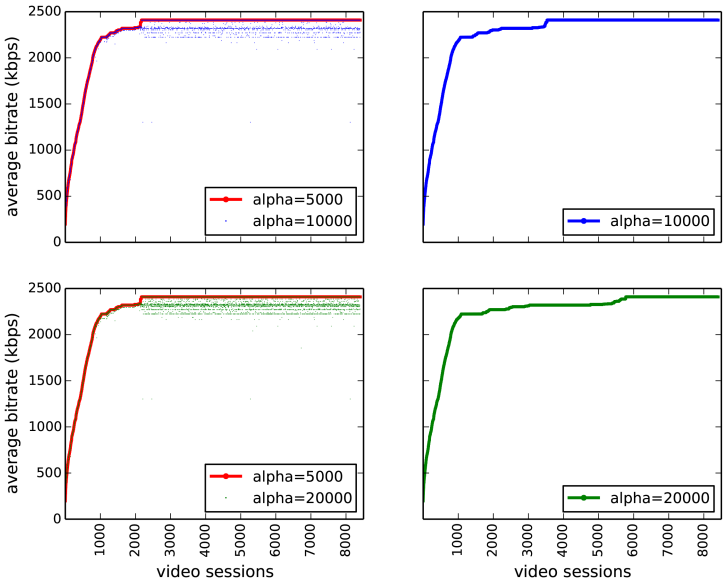

Fig. 11. Average bitrate from DP with different α values.

8.4 Gaining from Approximation

The performance guarantee provided in Theorem 7.1 for the greedy approach only expresses *the largest* gap its solution could possibly have from the optimal value. In general, this gap is much smaller. Among 8,408 sessions, 7,450 results from the greedy algorithm are optimal. Those not optimal are plotted in Figure 14. The figure shows how close the average bitrates obtained from

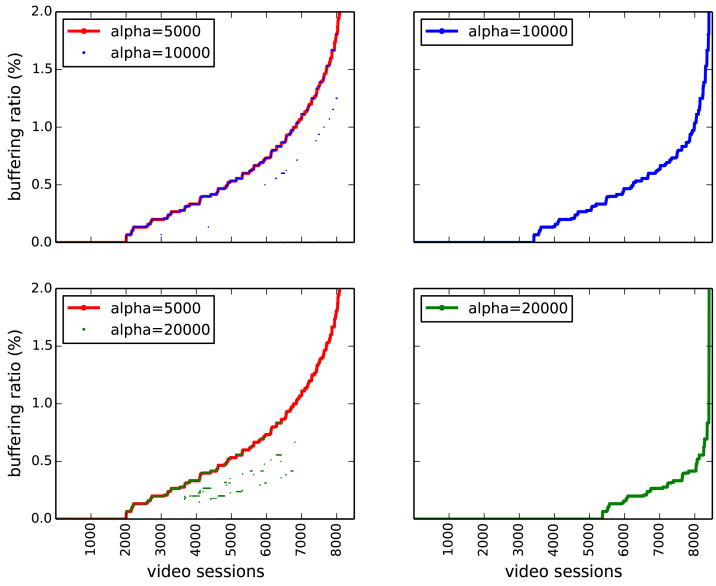

Fig. 12. Buffering ratio from DP with different α values.

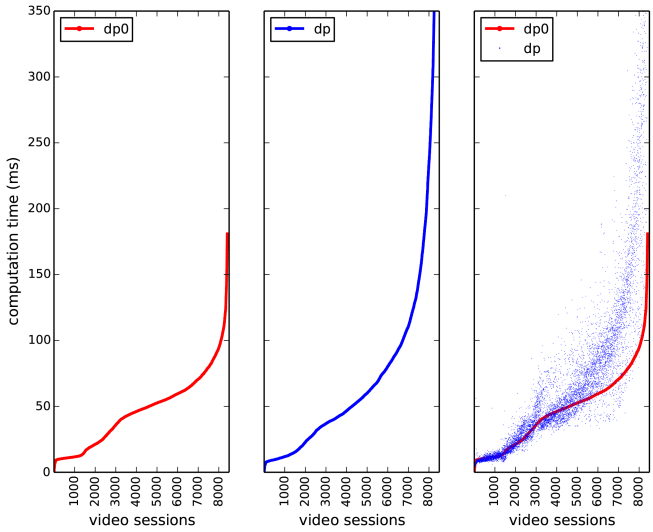

Fig. 13. Computation time of DP and DP0.

the greedy approach are from those obtained from DP0. The difference is in the range of 0 to 55. On average, the greedy approach gives an average bitrate of 2,204.60. Compared to 2,205.97 from the optimal solution, this is not a significant difference (0.062%).

The difference in the computation time, on the other hand, is counted by at least three orders of magnitude. The computation time of DP0 is in the range of 0 to 180.99ms, with an average of 46.61ms (Figure 15). The computation time of the greedy algorithm is in the range of 0 to 0.053ms, with an average of 0.01126ms.

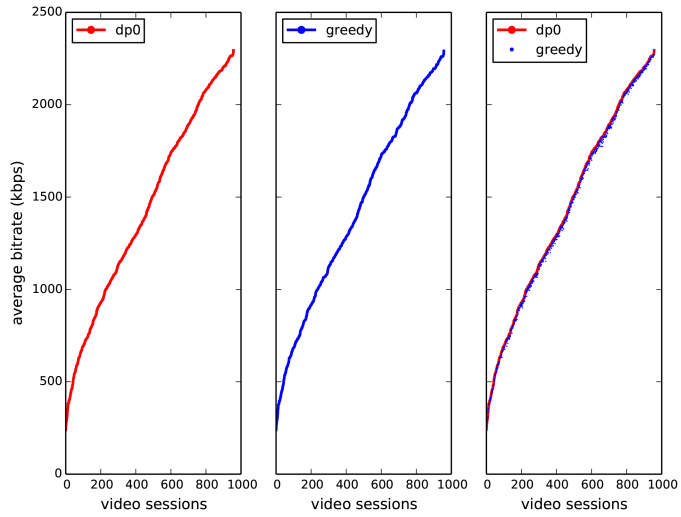

Fig. 14. Comparison of average bitrate between DP0 and greedy approach.

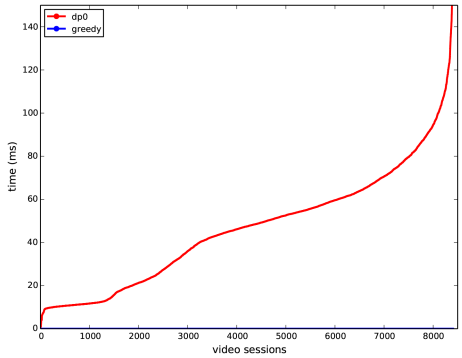

Fig. 15. Comparison of computation time between DP0 and greedy approach.

9 CONCLUSION AND FINAL REMARKS

The offline ABR optimization problems we considered are NP-hard and remain NP-hard in some special settings we have examined. At this point, dynamic programming offers a possible approach to the exact algorithms, where careful design is in demand to reduce the amount of space required for the execution. The proposed algorithms DP and DP0 provide some possible ways of doing so. The large volume of the data for analysis undoubtedly calls for algorithms that are both time and space efficient. To this end, our near-optimal greedy approach shows an excellent example with its striking performance.

Taking an initial step toward the benchmark of ABR solutions, we did not include in our study the effect of the buffer size: we only considered buffer sizes that are sufficiently large so that the download procedure never pauses. In doing so, we have left open the exploration of effective techniques to parameterize the buffer size.

Another factor not included in our study is the impact of the bitrate selection on the available bandwidth.

In regard to QoE, we have only considered three metrics: join time, average bitrate, and buffering ratio. Extending the study to other metrics also remains an interesting problem.

ACKNOWLEDGMENTS

The authors are in deep debt of gratitude to the anonymous peer reviewers for their insightful comments and constructive suggestions. The first author would like to extend her gratitude to the AMP lab at UC Berkeley for hosting her research work during her sabbatical leave from the University of Windsor.

REFERENCES

[1] Z. Akhtar, A. M. Le, Y. S. Nam, J. Chen, R. Govindan, E. Katz-Bassett, S. Rao, and J. Zhan. 2019. Improving adaptive video streaming through session classification. *ACM Journal of Data and Information Quality* 11, 4 (2019), 1–29.

[2] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. Ribeiro, J. Zhan, and H. Zhang. 2018. Oboe: Auto-tuning video ABR algorithms to network conditions. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM'18)*. Budapest, Hungary, 44–58.

[3] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and Hui Zhang. 2013. Developing a predictive model of quality of experience for internet video. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM'13)*. Hong Kong, China, 339–350.

[4] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann. 2018. A survey on bitrate adaptation schemes for streaming media over HTTP. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 562–585.

[5] F. Chiariotti, S. D'Aronco, L. Toni, and P. Frossard. 2016. Online learning adaptation strategy for DASH clients. In *Proceedings of the ACM Conference on Multimedia Systems (MMSys'16)*. Klagenfurt, Austria, 1–12.

[6] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. 2013. ELASTIC: A client-side controller for dynamic adaptive streaming over HTTP (DASH). In *Proceedings of the IEEE Packet Video Workshop (PV'13)*. San Jose, USA, 1–8.

[7] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. 2011. Understanding the impact of video quality on user engagement. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 362–373.

[8] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. 2019. Pano: Optimizing 360 video streaming with a better understanding of quality perception. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM'19)*. Beijing, China, 394–407.

[9] R. Houdaille and S. Gouache. 2012. Shaping HTTP adaptive streams for a better user experience. In *Proceedings of the 3rd ACM Conference on Multimedia Systems (MMSys'12)*. Chapel Hill, USA, 1–9.

[10] T. Y. Huang, A. J. Berglund, C. Ekanadham, and Z. Li. 2019. Hindsight: Evaluate video bitrate adaptation at scale. In *Proceedings of the 10th ACM Multimedia Systems Conference (MMSys'19)*. Amherst Massachusetts, USA, 86–97.

[11] T. Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. 2014. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4 (Oct. 2014), 187–198.

[12] J. Jiang, V. Sekar, H. Milner, D. Shepherd, I. Stoica, and H. Zhang. 2016. CFA: A practical prediction system for video QoE optimization. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*. Santa Clara, USA, 137–150.

[13] J. Jiang, V. Sekar, and H. Zhang. 2012. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT'12)*. Nice, France, 97–108.

[14] B. Korte and J. Vygen. 2008. *Algorithms and Combinatorics*. Combinatorial Optimization, Vol. 21. Springer.

[15] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. Begen, and D. Oran. 2014. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE Journal on Selected Areas in Communications* 32, 4 (April 2014), 719–733.

[16] Y. Liu, S. Dey, F. Ulupinar, M. Luby, and Y. Mao. 2015. Deriving and validating user experience model for DASH video streaming. *IEEE Transactions on Broadcasting* 61, 4 (2015), 651–665.

[17] H. Mao, R. Netravali, and M. Alizadeh. 2017. Neural adaptive video streaming with Pensieve. In *Proceedings of the ACM SIGCOMM*.

[18] V. Martin, J. Cabrera, and N. Garcia. 2016. Design, optimization and evaluation of a Q-learning HTTP adaptive streaming client. *IEEE Transactions on Consumer Electronics* 62, 4 (2016), 380–388.

[19] K. Miller, N. Corda, S. Argyropoulos, A. Raake, and A. Wolisz. 2013. Optimal adaptation trajectories for block-request adaptive video streaming. In *Proceedings of the IEEE Packet Video Workshop (PV'13)*. San Jose, USA, 1–8.

[20] R. K. P. Mok, E. W. M. Chan, and R. K. C. Chang. 2011. Measuring the quality of experience of HTTP video streaming. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management and Workshops (IM'11)*. Dublin, Ireland, 485–492.

[21] V. Nathan, V. Sivaraman, R. Addanki, M. Khani, P. Goyal, and M. Alizadeh. 2019. End-to-end transport for video QoE fairness. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM'19)*. Beijing, China, 408–423.

[22] D. Ray, J. Kosaian, K. V. Rashmi, and S. Seshan. 2019. Vantage: Optimizing video upload for time-shifted viewing of social live streams. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM'19)*. Beijing, China, 380–393.

[23] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. 2020. BOLA: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions on Networking* 28, 4 (2020), 1698–1711.

[24] Y. Sun, Yin X, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli. 2016. CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the ACM SIGCOMM*. 272–285.

[25] G. Tian and Y. Liu. 2012. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT'12)*. Nice, France, 109–120.

[26] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM'15)*. London, UK, 325–338.

[27] A. H. Zahran, J. Quinlan, D. Raca, C. J. Sreenan, E. Halepovic, R. K. Sinha, R. Jana, and V. Gopalakrishnan. 2016. OSCAR: An optimized stall-cautious adaptive bitrate streaming algorithm for mobile networks. In *Proceedings of ACM International Workshop on Mobile Video (MoVid'16)*. Klagenfurt, Austria, 1–6.

Received December 2020; revised March 2021; accepted May 2021