Algorithm Design Based on BOLA-Basic (2016)

Introduction

This algorithm is designed based on the BOLA algorithm introduced in 2016, specifically focusing on the BOLA-Basic approach. The principle of this algorithm is to maximize time-average utility by selecting a chunk size that maximizes this utility, thereby enhancing the overall viewing experience for the user.

In the literature, the user's viewing experience is defined by utility, which consists of two components: V_N and γS_N. The first part, V_N, is defined in Equation (1), and the second part, S_N, is defined in Equation (2).

Time-Average Utility V_N

The time-average utility V_N is defined as:

$$\overline{V}_N \triangleq \frac{\mathbb{E}\left\{\sum_{k=1}^{K_N} \sum_{m=1}^{M} a_m(t_k)\, v_m\right\}}{\mathbb{E}\left\{T_{\text{end}}\right\}} \tag{1}$$

Utility Component S_N

The second component of utility, S_N, is defined as:

$$\overline{S}_N \triangleq \frac{Np}{\mathbb{E}\left\{T_{\text{end}}\right\}} = \frac{\mathbb{E}\left\{\sum_{k=1}^{K_N} \sum_{m=1}^{M} a_m(t_k)\, p\right\}}{\mathbb{E}\left\{T_{\text{end}}\right\}} \tag{2}$$

Indicator Variable Definition

For each slot k, the indicator variable $a_m(t_k)$ is defined as follows:

$$a_m(t_k) = \begin{cases} 1 & \text{if the player downloads a segment of bitrate index } m \text{ in slot } k, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

where T_{end} represents the total duration from the start of downloading the first chunk to the end of playing the last chunk.

Chunk Utility v_m

The chunk utility v_m is defined as:

$$v_m = \ln\left(\frac{S_m}{S_1}\right) \tag{4}$$

where: - S_m describes the size of the chunk, specifically the chunk size corresponding to the m-th bitrate in ascending order. - S_1 is the smallest chunk size.

Additionally, p represents the playback duration of each chunk.

Optimization Problem Formulation

The authors of the literature divide the entire video download process into K_N slots. Since, in some slots, the algorithm may decide not to download any bitrate chunk, we have $K_N > n$.

In each slot k, given the buffer level $Q(t_k)$ at the start of the slot, the algorithm makes a control decision by solving the following deterministic optimization problem. Define:

$$\rho(t_k, a(t_k)) = \begin{cases} 0 & \text{if } \sum_{m=1}^{M} a_m(t_k) = 0, \\ \frac{\sum_{m=1}^{M} a_m(t_k)(Vv_m + V\gamma p - Q(t_k))}{\sum_{m=1}^{M} a_m(t_k) S_m} & \text{otherwise.} \end{cases} \tag{5}$$

Then, determine $a(t_k)$ by solving the following optimization problem:

$$\text{Maximize: } \rho(t_k, a(t_k)) \tag{6}$$

$$\text{Subject to: } \sum_{m=1}^{M} a_m(t_k) \leq 1, \quad a_m(t_k) \in \{0, 1\}. \tag{7}$$

The constraints of this problem result in a very simple solution structure. Specifically, the optimal solution is given by:

1. If $Q(t_k) > V(v_m + \gamma p)$ for all $m \in \{1, 2, \ldots, M\}$, then the no-download option is chosen, i.e., $a_m(t_k) = 0$ for all m. Note that in this case $T_k = \Delta$. 2. Else, the optimal solution is to download the next segment at bitrate index m^* where m^* is the index that maximizes the ratio $\frac{Vv_{m^*} + V\gamma p - Q(t_k)}{S_{m^*}}$ among all m for which this ratio is positive.

where V and γ are constants, and $Q(t_k)$ represents the number of chunks in the current buffer.

Implementation

In this section, we present the pseudocode that applies the optimization conclusions derived above. This is a loop structure with N iterations. Each iteration i is dedicated to selecting the bitrate for the i-th video chunk.

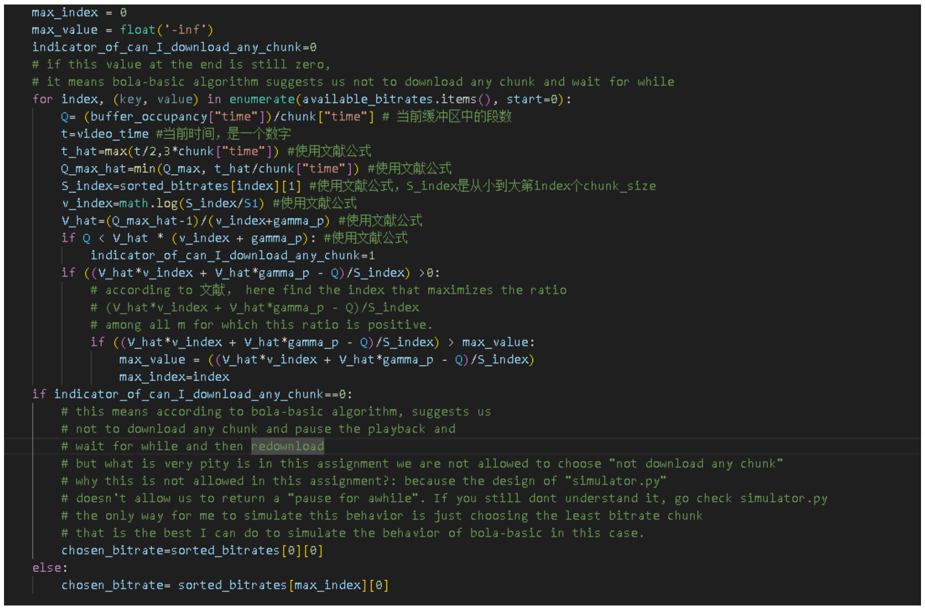
```
1:  for $n$ in $[1, N]$ do
2:      $t \leftarrow \min[\text{playtime from begin}, \text{playtime to end}]$
3:      $t' \leftarrow \max[t/2, 3p]$
4:      $Q^{\mathrm{D}}_{\max} \leftarrow \min[Q_{\max}, t'/p]$
5:      $V^{\mathrm{D}} \leftarrow (Q^{\mathrm{D}}_{\max} - 1)/(\upsilon_M + \gamma p)$
6:      $m^*[n] \leftarrow \arg\max_m (V^{\mathrm{D}}\upsilon_m + V^{\mathrm{D}}\gamma p - Q)/S_m$
7:      if $m^*[n] > m^*[n-1]$ then
8:          $r \leftarrow$ bandwidth measured when downloading segment $(n-1)$
9:          $m' \leftarrow \max m$ such that $S_m/p \leq \max[r, S_1/p]$
10:         if $m' \geq m^*[n]$ then
11:             $m' \leftarrow m^*[n]$
12:         else if $m' < m^*[n-1]$ then
13:             $m' \leftarrow m^*[n-1]$
14:         else if some utility sacrificed for fewer oscillations then
15:             pause until $(V^{\mathrm{D}}\upsilon_{m'} + V^{\mathrm{D}}\gamma p - Q)/S_{m'} \geq$
                        $(V^{\mathrm{D}}\upsilon_{m'+1} + V^{\mathrm{D}}\gamma p - Q)/S_{m'+1}$     ▷ BOLA-O
16:         else
17:             $m' \leftarrow m' + 1$                                            ▷ BOLA-U
18:         end if
19:         $m^*[n] \leftarrow m'$
20:     end if
21:     pause for $\max[p \cdot (Q - Q^{\mathrm{D}}_{\max} + 1), 0]$
22:     download segment $n$ at bitrate index $m^*[n]$, possibly abandoning
23: end for
```

Figure 1: Pseudocode

Each iteration generates a number k, representing the index of the bitrate (from smallest to largest) chosen for downloading the i-th video chunk. The sequence of selected indices forms a list $m'[n]$, where each entry in $m'[n]$ corresponds to the optimal bitrate index for each chunk. Lines 2-6 and 21-22 in this picture are the BOLA-Basic implementation, which I implemented, corresponding to the core of my `studentcode_120090155.py`.

Core Implementation in Python

The following Python code in `studentcode_120090155.py` implements the BOLA-Basic algorithm.



```python
max_index = 0
max_value = float('-inf')
indicator_of_can_I_download_any_chunk=0
# if this value at the end is still zero,
# it means bola-basic algorithm suggests us not to download any chunk and wait for while
for index, (key, value) in enumerate(available_bitrates.items(), start=0):
    Q= (buffer_occupancy["time"])/chunk["time"] # 当前缓冲区中的段数
    t=video_time #当前时间，是一个数字
    t_hat=max(t/2,3*chunk["time"]) #使用文献公式
    Q_max_hat=min(Q_max, t_hat/chunk["time"]) #使用文献公式
    S_index=sorted_bitrates[index][1] #使用文献公式，S_index是从小到大第index个chunk_size
    v_index=math.log(S_index/S1) #使用文献公式
    V_hat=(Q_max_hat-1)/(v_index+gamma_p) #使用文献公式
    if Q < V_hat * (v_index + gamma_p): #使用文献公式
        indicator_of_can_I_download_any_chunk=1
    if ((V_hat*v_index + V_hat*gamma_p - Q)/S_index) >0:
        # according to 文献， here find the index that maximizes the ratio
        # (V_hat*v_index + V_hat*gamma_p - Q)/S_index
        # among all m for which this ratio is positive.
        if ((V_hat*v_index + V_hat*gamma_p - Q)/S_index) > max_value:
            max_value = ((V_hat*v_index + V_hat*gamma_p - Q)/S_index)
            max_index=index
if indicator_of_can_I_download_any_chunk==0:
    # this means according to bola-basic algorithm, suggests us
    # not to download any chunk and pause the playback and
    # wait for while and then redownload
    # but what is very pity is in this assignment we are not allowed to choose "not download any chunk"
    # why this is not allowed in this assignment?: because the design of "simulator.py"
    # doesn't allow us to return a "pause for awhile". If you still dont understand it, go check simulator.py
    # the only way for me to simulate this behavior is just choosing the least bitrate chunk
    # that is the best I can do to simulate the behavior of bola-basic in this case.
    chosen_bitrate=sorted_bitrates[0][0]
else:
    chosen_bitrate= sorted_bitrates[max_index][0]
```

Figure 2: Python Code Implementation of BOLA-Basic Algorithm

Explanation of Core Code Logic

My code primarily implements lines 2-6 of the pseudocode in the figure. Observers can easily connect these lines of code to lines 2-6 of the pseudocode, with the only difference being the use of different variable names. The main purpose of this code is to find m', where m^* is the index that maximizes the ratio $\frac{Vv_{m^*}+V\gamma p-Q(t_k)}{S_{m^*}}$ among all m for which this ratio is positive. It is important to note, as commented in my code:

```
if indicator_of_can_I_download_any_chunk == 0:
    # this means according to bola-basic algorithm, suggests us
    # not to download any chunk and pause the playback and
    # wait for a while and then redownload
    # but what is very pity is in this assignment we are not allowed
    # to choose "not download any chunk"
    # why this is not allowed in this assignment?:
    # because the design of "simulator.py"
    # doesn't allow us to return a "pause for awhile".
    # If you still don't understand it, go check simulator.py.
    # the only way for me to simulate this behavior
    # is just choosing the least bitrate chunk
    # that is the best I can do to simulate the behavior of bola-basic in this case.
```

The reason the main objective is to find m', where m^* is the index that maximizes the ratio $\frac{Vv_{m^*}+V\gamma p-Q(t_k)}{S_{m^*}}$ among all m for which this ratio is positive, can be understood from the mathematical principles we discussed earlier. Similarly, the rationale for choosing not to download can also be found here: If $Q(t_k) > V(v_m + \gamma p)$ for all $m \in \{1, 2, \ldots, M\}$, then the no-download option is chosen.Lines 21-22 in the pseudocode also indicate that, under these conditions, we should not select any chunk. However, due to the design of our project, I am required to return a bitrate, which prevents me from fully implementing the BOLA-Basic algorithm as intended.

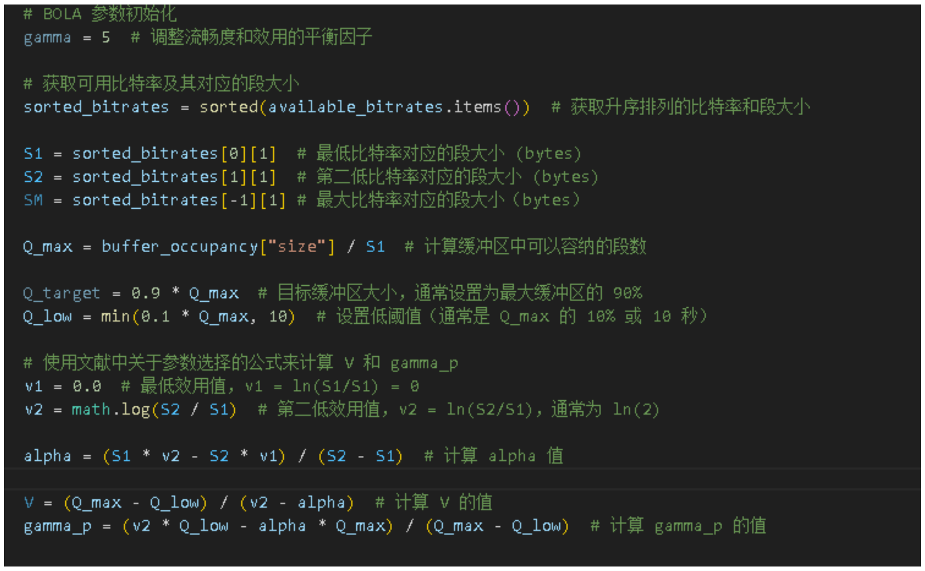
Parameter Calculation for V and γp

To complete the implementation of lines 2-6 in the pseudocode, we need to determine the parameters V and γp. The authors provided the following method to find these parameters:

$$V = \frac{Q_{\max} - Q_{\text{low}}}{v_M - \alpha}, \quad \gamma p = \frac{v_M Q_{\text{low}} - \alpha Q_{\max}}{Q_{\max} - Q_{\text{low}}} \tag{8}$$

where

$$\alpha = \frac{S_1 v_2 - S_2 v_1}{S_2 - S_1}. \tag{9}$$



```
# BOLA 参数初始化
gamma = 5  # 调整流畅度和效用的平衡因子

# 获取可用比特率及其对应的段大小
sorted_bitrates = sorted(available_bitrates.items())  # 获取升序排列的比特率和段大小

S1 = sorted_bitrates[0][1]  # 最低比特率对应的段大小 (bytes)
S2 = sorted_bitrates[1][1]  # 第二低比特率对应的段大小 (bytes)
SM = sorted_bitrates[-1][1] # 最大比特率对应的段大小（bytes）

Q_max = buffer_occupancy["size"] / S1  # 计算缓冲区中可以容纳的段数

Q_target = 0.9 * Q_max  # 目标缓冲区大小，通常设置为最大缓冲区的 90%
Q_low = min(0.1 * Q_max, 10)  # 设置低阈值（通常是 Q_max 的 10% 或 10 秒）

# 使用文献中关于参数选择的公式来计算 V 和 gamma_p
v1 = 0.0  # 最低效用值, v1 = ln(S1/S1) = 0
v2 = math.log(S2 / S1)  # 第二低效用值, v2 = ln(S2/S1), 通常为 ln(2)

alpha = (S1 * v2 - S2 * v1) / (S2 - S1)  # 计算 alpha 值

V = (Q_max - Q_low) / (v2 - alpha)  # 计算 V 的值
gamma_p = (v2 * Q_low - alpha * Q_max) / (Q_max - Q_low)  # 计算 gamma_p 的值
```

Figure 3: Parameter Choice

In this implementation: - S_1 and S_2 represent the chunk sizes corresponding to the lowest and second-lowest bitrates. - v_1 and v_2 are the utility values, with $v_1 = 0$ (since $v_1 = \ln(S_1/S_1) = 0$) and $v_2 = \ln(S_2/S_1)$. - $Q_{\max}$ is the maximum buffer capacity in terms of the number of chunks, and Q_{low} is a low threshold, typically set to 10% of $Q_{\max}$ or 10 seconds.

This initialization ensures that the parameters V and γp are correctly set to allow the BOLA algorithm to optimize video streaming quality and smoothness.

Modified BOLA-Basic Algorithm

In addition to the original BOLA-Basic implementation, I created a modified version of the algorithm to better suit the constraints of our assignment. In this alternative implementation, I determined the parameters V and γ through a series of experiments to achieve improved scores in `grader.py`.

Buffer Stability and Bitrate Switching

To avoid frequent and unnecessary bitrate switching, I introduced a condition that only downgrades the bitrate when the buffer is low and the previous throughput was insufficient to support the current bitrate. This helps maintain a stable viewing experience. The code snippet below implements this logic:

```
# Current buffer state
buffer_level = buffer_occupancy["time"] / chunk["time"]
```

```
# Prevent bitrate downgrades unless buffer is low and
# previous throughput was lower than the chosen bitrate
if prev_throughput < chosen_bitrate and buffer_level < Q_low:
    chosen_bitrate = sorted_bitrates[0][0]
```

This modification ensures that bitrate downgrades occur only when necessary, avoiding frequent changes in quality and thereby enhancing stability.

Key Differences from the Original BOLA-Basic

In this modified implementation, I chose to depart from the original BOLA-Basic algorithm's direct optimization formula. Instead, I focused on its core principle:

> The principle of this algorithm is to maximize time-average utility by selecting a chunk size that maximizes this utility, thereby enhancing the overall viewing experience for the user.

The literature defines the user's viewing experience in terms of a utility function, which has two components: V_N and γS_N. However, in our project environment, it was challenging to fully implement the BOLA algorithm due to assignment restrictions. Specifically, the assignment does not allow us to skip downloading a chunk and wait, which is sometimes required by the BOLA algorithm to balance buffer levels and streaming quality.

Another limitation was accurately approximating $Q_{\max}$, which ideally represents the maximum buffer capacity in terms of the number of chunks. Due to the lack of comprehensive parameters in the assignment, I approximated $Q_{\max}$ as follows:

$$Q_{\max} = \frac{\text{buffer\_occupancy["size"]}}{S_1} \tag{10}$$

where S_1 is the chunk size for the lowest bitrate. This approximation is crude and does not accurately reflect the actual buffer capacity, limiting its effectiveness in optimizing the BOLA algorithm.

Maximizing the Objective Function

Given these constraints, directly applying the mathematical conclusions from the original BOLA-Basic optimization formula proved impractical and resulted in poor scores in `grader.py`. Therefore, instead of strictly following the original optimization, I focused on maximizing the objective function by selecting the chunk size that maximizes $V_N + \gamma S_N$, which aligns with the BOLA algorithm's intent to optimize viewing experience.

To implement this approach, I created a new file, `studentcode_120090155_1.py`, which simplifies the algorithm to select the chunk size that maximizes $V_N + \gamma S_N$. This adjustment allows the algorithm to provide a better viewing experience within the limitations of our project environment.

A comparison between the pre-improvement BOLA-BASIC implementation and the post-improvement BOLA-BASIC implementation shows that the pre-improvement version performs significantly worse in several test cases, particularly in terms of average bitrate, switching count, and overall score. Let us analyze the reasons why the pre-improvement BOLA-BASIC implementation might be inferior compared to the post-improvement one.

1. Summary of Performance Comparison

The pre-improvement implementation generally performs poorly across most tests, particularly in `badtest` and `testALThard`, resulting in low average bitrates, higher switching counts, and lower scores. On the other hand, the post-improvement implementation consistently achieves higher average bitrates, fewer switches, and better overall scores, performing particularly well in the `testALTsoft` and `testHD` scenarios.

Below, we provide specific comparisons and potential reasons for these differences:

2. Key Problems and Hypotheses

A. Switching Count

The pre-improvement implementation results in significantly more switching: notably, in the `badtest` and `testALThard` tests, the pre-improvement version had 18 and 14 switches respectively, while the post-improvement version had only 6 switches in the same tests. This is a notable difference.

Possible Causes:

- **Insufficient Buffer Safety Factor Adjustment:** The pre-improvement implementation may have insufficient adjustments to the buffer safety factor when calculating utility, leading to frequent bitrate changes.
- **Lack of Constraints on Frequent Switching:** The post-improvement implementation prevents frequent bitrate changes by reducing bitrate when previous throughput is lower than the chosen bitrate. In the pre-improvement implementation, despite having a similar mechanism, the logic might not be working effectively to prevent frequent switching.

B. Average Bitrate

The average bitrate of the pre-improvement implementation is lower: in the `badtest` and `testALThard` tests, the pre-improvement average bitrate is only 700,000 bps, whereas the post-improvement version achieves 950,000 bps.

Possible Causes:

- **Overly Conservative Bitrate Selection Strategy:** The pre-improvement implementation may be too conservative in selecting the bitrate, tending to choose a lower bitrate, especially when buffer occupancy is low or insufficient, which leads to a lower average bitrate.
- **Adjustments of V and γ:** The post-improvement version may have chosen more suitable V and γ parameters, which directly impact the balance of utility calculations and consequently affect bitrate selection. The pre-improvement version may have too small a V, which causes an imbalance between utility and the safety factor, leading to inadequate bitrate enhancement.

3. Major Limitations and Improvements

The main issues in the pre-improvement implementation are due to constraints imposed by the assignment, which conflict with the intended BOLA algorithm behavior. The following two points are particularly important:

- **Unable to Skip Chunk Download and Wait:** The assignment restriction does not allow us to skip downloading a chunk and wait, which is sometimes required by the BOLA algorithm to balance buffer levels and streaming quality. The BOLA algorithm may, under certain conditions, suggest skipping the download of specific chunks and waiting until the network or buffer conditions improve to maintain a better playback experience. However, due to the simulator constraints in this assignment, we are forced to choose a bitrate to download the current chunk, making it impossible to implement the ”pause and wait” logic described in the algorithm. This limitation could lead the implementation to select the lowest bitrate when buffer levels are low, instead of waiting for a better opportunity.
- **Approximation of $Q_{\max}$:** Another limitation is accurately approximating $Q_{\max}$, which ideally represents the maximum buffer capacity in terms of the number of chunks. Due to the lack of comprehensive parameters in the assignment, I approximated $Q_{\max}$ as follows:

  $$Q_{\max} = \frac{\text{buffer\_size}}{S_1}$$

  where buffer_size is the total buffer capacity in bytes, and S_1 is the segment size corresponding to the lowest bitrate (in bytes). This approximation may not be precise enough, as in real-world applications, $Q_{\max}$ should take into account dynamic network conditions and variations in video segment bitrates.

These limitations have resulted in the pre-improvement BOLA-BASIC algorithm underperforming in some scenarios, particularly

in low-buffer and high-variance network environments. In the post-improvement version, adjustments to the V and γ parameters, as well as adopting a more appropriate bitrate selection strategy, have led to better performance given the constraints of the assignment.