Task 1: Adaptation Algorithm for Adaptive Streaming over HTTP

Algorithm Analysis:

This algorithm assigns the highest priority to decreasing the buffer time. It also tries to minimize the quality shifts and enhances the minimum and average video quality. It also decreases the startup time before the playback at the beginning. The outputs of this algorithm include the delay time of downloading B_{delay} and the bitrate of the next chunk $r_{n(t)+1}$.

The inputs include some tuples providing the former chosen chunks information, including their bitrate, the time when the chunks being downloaded and the time when the chunks being download completely the next chunk information including available bitrates and corresponding chunk size, the bitrate of the previous chunk $\tilde{\rho}_{n(t)}$ and $\sigma_i = (r_i, t_i^b, t_i^e)$ the buffer level at time t $\beta_{(t)}$.

The algorithm can be configured with following parameters: The buffer level B_{min}, B_{low} B_{high} measured in seconds of playback (B_{min}, the buffer level where the minimum bitrate must be chosen to avoid playback interruption; B_{low} and B_{high} indicate the boundary of the target interval); time duration Δ_t to compute previous average bitrate.

The algorithm can be separated into 3 phases.

1) Initialization

This is when the algorithm works for the first time when the user requests for the video. To decrease the startup time, the algorithm downloads the first chunk at the lowest bitrate but later increases video quality in an aggressive way.

Algorithm 1: ADAPTATION ALGORITHM

Input: $(\sigma_i)_{i=1,\dots,n(t)}$
Output: $r_{n(t)+1}$, B_{delay}

static runningFastStart := true;
$B_{\text{delay}} := 0$;
$r_{n(t)+1} := r_{n(t)}$;
if runningFastStart ...
  $\wedge\ r_{n(t)} \neq r_{\max}$ ...
  $\wedge\ \beta_{\min}(t_1) \leq \beta_{\min}(t_2)\, \forall t_1 < t_2 \leq t$ ...
  $\wedge\ r_{n(t)} \leq \alpha_1 \cdot \tilde{\rho}(t - \Delta_t, t)$ **then**
    if $\beta(t) < B_{\text{min}}$ **then**
      if $r^{\uparrow}_{n(t)} \leq \alpha_2 \cdot \tilde{\rho}(t - \Delta_t, t)$ **then**
        $r_{n(t)+1} := r^{\uparrow}_{n(t)}$;
    else if $\beta(t) < B_{\text{low}}$ **then**
      if $r^{\uparrow}_{n(t)} \leq \alpha_3 \cdot \tilde{\rho}(t - \Delta_t, t)$ **then**
        $r_{n(t)+1} := r^{\uparrow}_{n(t)}$;
    else
      if $r^{\uparrow}_{n(t)} \leq \alpha_4 \cdot \tilde{\rho}(t - \Delta_t, t)$ **then**
        $r_{n(t)+1} := r^{\uparrow}_{n(t)}$;
      if $\beta(t) > B_{\text{high}}$ **then**
        $B_{\text{delay}} := B_{\text{high}} - \tau$;
else
  runningFastStart := false;
  if $\beta(t) < B_{\text{min}}$ **then**
    $r_{n(t)+1} := r_{\min}$;
  else if $\beta(t) < B_{\text{low}}$ **then**
    if $r_{n(t)} \neq r_{\min} \wedge r_{n(t)} \geq \tilde{\rho}_{n(t)}$ **then**
      $r_{n(t)+1} := r^{\downarrow}_{n(t)}$;
  else if $\beta(t) < B_{\text{high}}$ **then**
    if $r_{n(t)} = r_{\max} \vee r^{\uparrow}_{n(t)} \geq \alpha_5 \cdot \tilde{\rho}(t - \Delta_t, t)$ **then**
      $B_{\text{delay}} := \max(\beta(t) - \tau,\ B_{\text{opt}})$;
  else
    if $r_{n(t)} = r_{\max} \vee r^{\uparrow}_{n(t)} \geq \alpha_5 \cdot \tilde{\rho}(t - \Delta_t, t)$ **then**
      $B_{\text{delay}} := \max(\beta(t) - \tau,\ B_{\text{opt}})$;
    else
      $r_{n(t)+1} := r^{\uparrow}_{n(t)}$;

2) Fast start phase

There are three requirements of this phase. The first is a flag $runningFastStart$. The second is the minimum buffer level within very short period and the last is a judgement of whether the next chunk size in larger bitrate is smaller than a certain percentage of previous average bitrate, where α_1 set the percentage value. If all three conditions are satisfied, the video quality will ramp up at aggressively or will be maintained if the buffer level is too high.

3) Steady phase

If one of the conditions of above fails, then it will skip to the steady phase. In this phase, the bitrate is stabilized by setting delay time to decrease video quality shift or the bitrate will be decreased if the network condition is poor.

Implementation:

In the simulator.py, changes are made to fulfill the delay mechanism

```
if Bdelay==0 or buffer.time<=Bdelay:
        logger.log_bitrate_choice(current_time, chunknum, (chosen_bitrate, stu_chunk_size))

#simulate download and playback
        time_elapsed = trace.simulate_download_from_time(current_time, stu_chunk_size)


#round time to remove floating point errors
#todo: this did not fix them
        time_elapsed = round(time_elapsed, 3)

        if len(previous_list)>=5:
            previous_list.pop(4)
            previous_list.insert(0,(buffer.time,stu_chunk_size,current_time,current_time+time_elapsed))
        else:
            previous_list.insert(0,(buffer.time,stu_chunk_size,current_time,current_time+time_elapsed))

        rebuff_time = buffer.sim_chunk_download(stu_chunk_size, chunk_arg["time"], time_elapsed)


#update state variables
        prev_throughput = (stu_chunk_size * 8) / time_elapsed
        current_time += time_elapsed
        chunks_remaining -= 1
```

```
#get next chunk
        chunknum, chunk = next(chunk_iter, (None, None))
else:
    current_time += .5
    buffer.burn_time(.5)
    continue
```

```
Bmin=10
Bmax=30
Blow=20
alpha1=0.33
alpha2=0.3
alpha3=0.4
alpha4=0.5
alpha5=0.65
runningFastStart=True
```

In the studentcodeExample.py
The configuration is provided here.
An *Average_bit_rate_calculation* function is used to calculate the average bitrate of a duration of time $\Delta_t = 10s$. The chunk size is 2s, so the previous list contains the size of 5 previous chunks and their download time. The *student_entrypoint* function goes as the flow of the algorithm. (See in the code file)

```
def Average_bit_rate_calculation(previous_list):
    if len(previous_list)==0:
        return 0
    else:
        download_time=0
        total_size=0
        for i in range(len(previous_list)):
            download_time+=(previous_list[i][3]-previous_list[i][2])
            total_size+=previous_list[i][1]*8
        return (total_size/download_time)
```

Evaluation:
By using the grader.py, the score is shown here. The first picture is the performance of the algorithm in this paper and the second one is the benchmark algorithm. Compared to the buffer-based example, this algorithm significantly decreases the buffer time and decreases the number of bitrate switch to ensure the video quality. The average bitrate can be altered by using different arguments but has limited improvement. The disadvantage in the average bitrate might results in that the algorithm takes reducing the buffer time as its first priority and the grading criteria cannot support this algorithm well.

```
testALThard:
Results:Average bitrate:1583333.3333333333buffer time:1.8889999999999998switches:5
Score:947177.5198363662

testHD:
Results:Average bitrate:2333333.3333333335buffer time:0.101switches:5
Score:1529910.4326800974


testPQ:
Results:Average bitrate:500000.0buffer time:91.115switches:0
Score:4669.348620686024


badtest:
Your trace file is poorly formed!Results:Average bitrate:1583333.3333333333buffer time:1.8889999999999998switches:5
Score:947177.5198363662


testALTsoft:
Results:Average bitrate:1750000.0buffer time:0.101switches:7
Score:971187.1426653258


testHDmanPQtrace:
Results:Average bitrate:500000.0buffer time:91.115switches:0
Score:4669.348620686024
```

```
testALThard:
Results:Average bitrate:2600000.0buffer time:19.751switches:28
Score:91422.15235967688


testHD:
Results:Average bitrate:4700000.0buffer time:0.101switches:1
Score:4301656.912826439


testPQ:
Results:Average bitrate:500000.0buffer time:91.115switches:0
Score:4669.348620686024


badtest:
Your trace file is poorly formed!Results:Average bitrate:2600000.0buffer time:19.751switches:28
Score:91422.15235967688


testALTsoft:
Results:Average bitrate:4100000.0buffer time:5.948000000000001switches:9
Score:1426836.8172498895


testHDmanPQtrace:
Results:Average bitrate:500000.0buffer time:91.115switches:0
Score:4669.348620686024
```