Ensemble Adaptive Streaming – A New Paradigm to Generate Streaming Algorithms via Specializations

Guanghui Zhang and Jack Y. B. Lee, *Senior Member*, IEEE

Abstract—Video streaming is now ubiquitous in the mobile Internet. This motivated intense research in adaptive streaming algorithms to tackle mobile networks' fluctuating conditions. Our investigations revealed that while existing algorithms can perform well in their intended operating environments, their performance can degrade substantially in other environments. This work tackles this challenge by developing a novel Ensemble Adaptive Streaming (EAS) paradigm to mobile video streaming. As opposed to designing a single streaming algorithm for all network conditions, we argue that different network conditions require different *algorithms*. We introduce the notion of *network differentiators* to segregate network conditions into different classes where each class has its own adaptation algorithm designed and optimized specifically for it. An EAS mobile streaming client then selects at runtime the matching adaptation algorithm using the same network differentiators on a per session basis for streaming. We show how EAS can be applied to existing machine-learning approaches to improve their performances. Moreover, to fully exploit EAS's potential we developed the first Genetic Programming approach to evolve adaptive streaming algorithms. The resultant EAS-GP algorithms not only outperformed state-of-the-art algorithms substantially, but also exhibited remarkable robustness over time, location, mobile operators, as well as quality-of-experience metrics.

Index Terms— Video Streaming; Mobile Network; Genetic Programming; Quality-of-experience.

1 Introduction

With the rapid advances in high-speed mobile networks such as 4G/LTE [1] and the upcoming 5G [2], streaming video has become an essential application for mobile users. According to an on-going forecast [3], streaming video is poised to account for 75% of all mobile data traffic by 2020.

Given streaming video's ubiquitous importance its quality-of-experience (QoE) has become a grand challenge to content and service providers. Unlike wired networks where bandwidth is relatively abundant and consistent, mobile bandwidth is highly variable depending on a variety of factors such as coverage, mobility, user density, and so on. Therefore, it is now standard practice for content providers to deploy adaptive streaming systems to automatically adjust the video bitrate in accordance to the network bandwidth available. This has led to the recent standardization of adaptive video streaming framework by MPEG – known as MPEG-DASH [4], for wider inter-operability across different adaptive streaming systems.

The heart of an adaptive video streaming system is its adaptation logic or algorithm. Using information collected during runtime, e.g., estimated throughput, client buffer occupancy, etc., the algorithm then determines the best bitrate for future video segments. Different adaptation logics differ in: (a) the types of parameters adopted in bitrate selection; (b) the metrics they used for performance optimization; and (c) the tradeoffs between different metrics. These could either be explicitly designed into the algorithm or an implicit behavior of the algorithm.

Researchers have developed a wide variety of adaptive streaming algorithms in recent years (e.g., [5-13]). These have all shown significant QoE improvements compared to their non-adaptive counterparts. However, our investigations revealed that while existing algorithms can perform well in their intended operating environments, their performances often degrade substantially in other environments. This work proposes a novel Ensemble Adaptive Streaming (EAS) paradigm to tackle this challenge. As opposed to designing a *single* universal streaming algorithm for all network environments, we argue that different networks require *different algorithms*. We introduce the notion of *network differentiators* to segregate networks into different classes where each class has its own adaptation algorithm designed and optimized specifically for it. An EAS mobile streaming client then selects *at runtime* the matching adaptation algorithm using the same network differentiators for use in each streaming session.

This work has four main contributions. First, our extensive experiments confirmed that it is fundamentally difficult, if not impossible, for a single adaptation algorithm, even state-of-the-art ones such as MPC [5] and Pensieve [6], to perform equally well across a wide range of network conditions. Moreover, our investigations also showed that network conditions can and do vary significantly on a *daily* basis, even for the *same* location with no mobility. Thus operating over a wide range of network conditions is not an option but a necessity in practice.

Our second contribution is in developing the novel Ensemble Adaptive Streaming paradigm to tackle the abovementioned challenge. We introduce the notion of network

• *The authors are with the Department of Information Engineering at the Chinese University of Hong Kong, Shatin, NT, Hong Kong SAR. E-mail: {ghzhang, yblee}@ie.cuhk.edu.hk*

differentiators as a mean to segregate networks into separate classes, and by designing specialized algorithms for each class one can achieve significant performance gains over existing approaches.

Third, through applying EAS to Pensieve we discovered that although EAS-Pensieve can achieve substantially better performance, its efficacy was still restricted under challenging network conditions, presumably due to the inherent structure of the neural network model employed. To tackle this limitation we developed, for the first time in the literature, a Genetic Programming (GP) [14] approach to evolve specialized adaptation algorithms for EAS. This EAS-GP approach not only substantially outperformed current state-of-the-art algorithms, but also exhibited remarkable robustness over time, locations, mobile operators, and even different QoE metrics. Moreover, EAS-GP can be deployed with only minimal changes to existing streaming players, making it readily deployable.

Last but not least, we developed a new rate-throughput-buffer (RTB) analysis tool that can offer new insights into the inner-working of various adaptation algorithms. For example, it revealed the reason behind Pensieve's significant performance variations under challenging network conditions; and showed why it is fundamentally difficult for a single algorithm to perform well across a wide range of network conditions.

The rest of the paper is organized as follows: Section 2 reviews some previous related works; Section 3 demonstrates the limitations of solo adaptive streaming; Section 4 presents the EAS paradigm and demonstrates its potential by applying it to Pensieve; Section 5 presents a Genetic Programming approach to evolving specialized adaptation algorithms for EAS; Section 6 compares performance of EAS against the state-of-the-arts; and Section 7 summarizes the study and outlines some future work.

2 Related Work

In recent years, video streaming protocols such as Microsoft Smooth-Streaming [15], Apple's HLS [16] and Adobe's HDS [17] have nearly all migrated to HTTP-based adaptive streaming. This class of protocols is being standardized under the umbrella of Dynamic Adaptive Streaming over HTTP (i.e., DASH) [4]. A detailed review of existing works is beyond the scope of this work. We refer the interested readers to the studies by Seufert *et al*. [18], Juluri *et al*. [19], Kua *et al*. [20] and Bentaleb *et al.* [21] for survey and comparison of existing streaming algorithms. In the following we briefly review some related studies.

Built-upon intuitions, researchers developed numerous adaptive streaming algorithms over the years. For example, Spiteri *et al.* [7] devised an online control algorithm called BOLA that employed Lyapunov optimization techniques to adapt the video bitrate; Liu *et al.* [8] proposed a data-driven framework called PSRA where past throughput trace data are analyzed to automatically optimize streaming parameters according to the underlying network and system configurations; Zahran *et al*. [9] proposed ARBITER+ which integrated several tunable components to ensure high video QoE; Qin *et al.* [10] designed an algorithm that leveraged PID feedback control concepts to track a target buffer occupancy to prevent playback rebuffering; and so on.

In another study Yin *et al.* [5] proposed Robust-MPC which makes bitrate decisions by solving a QoE maximization problem over a horizon of several future segments. By optimizing directly for a desired QoE objective, Robust-MPC can perform better than approaches that employed fixed heuristics. However, Robust-MPC was not tuned for different network environments and relies heavily on accurate throughput estimation which are not always available. When throughput predictions are incorrect, Robust-MPC's performance can degrade significantly [22].

Another approach to the design of adaptation algorithms is based on machine learning techniques. Generally, in a separate training phase the system is exposed to the target operating environment where it makes bitrate decisions solely through observations of the resulting performance of past experiences. One approach is to model the bitrate adaptation problem as a Markov decision process (MDP) and then employ machine learning algorithms to solve it. For example, applying reinforcement learning (RL) in a tabular form (e.g., Q-learning [23]) is a common solution [11-13]. It represents the model to be learned as a table, with separate entries for all states (e.g., estimated throughput, client buffer occupancy, etc.) as well as bitrate decisions. The challenge lies in the choice of state space. A large state space can more accurately capture the environment's characteristics albeit with tradeoffs in complexity and convergence time [24].

In another study, Mao *et al.* proposed Pensieve [6] - a system to generate adaptive streaming algorithms using A3C - a deep RL algorithm [25], through training a neural network for bitrate adaptation. Pensieve was shown to outperform many state-of-the-art adaptation algorithms. Most recently, Akhtar *et al.* [38] proposed Oboe that pre-computes, for a given adaptive streaming algorithm, the best parameters for different network conditions, and then dynamically adjusts the parameters at run-time according to the changing network condition.

Regardless of the approaches, current algorithms were all *solo*, i.e., a single optimized algorithm is to be used for all settings. We argue that this is sub-optimal as variations in network environments not only require adaptation of the video bitrate, but also *adaptation of the adaptation algorithm itself*. We establish this argument in the next section.

3 Solo Adaptive Streaming

In this section we demonstrate the limitations of solo adaptive streaming.

3.1 Experiment Setup

To evaluate streaming algorithms in realistic network settings we employed trace-driven simulations where the simulator implemented the adaptive streaming algorithms and executed them using throughput trace data obtained from real-world production mobile networks.

Table 1 - Information of Throughput Trace Datasets.

Characteristics	Dataset				
	#1	#2	#3	#4	Overall
Mean Throughput (Mbps)	5.57	4.71	3.29	2.87	4.01
Min/max Session Throughput (Mbps)	0.20 / 10.44	0.38 / 8.34	0.09 / 9.78	0.22 / 7.84	0.09 / 10.44
Coefficient of Variation	0.44	0.39	0.74	0.53	0.53
Location	L1	L1	L2	L3	n/a
Service Provider	S1	S2	S1	S1	n/a

Table 2 - System Settings.

Parameters	Values
Bitrate Profile	[0.2, 0.4, 0.8, 1.2, 2.2, 3.3, 5.0, 6.5, 8.6, 10.0, 12.0] Mbps
Segment Duration	2 s
Video Duration	Empirical distribution (40 s to 600 s)
Prefetch Duration	10 s
Startup Bitrate	200 kbps
Client Buffer Size	60 s

Table 3 - Normalized QoE Performance (%) and Coefficient of Variation (CoV) for Solo-adaptive Streaming.

Algorithm	Dataset					
	#1	#2	#3	#4	Overall QoE	Overall CoV
Pensieve	79.2	79.7	56.3	67.0	72.5	0.91
MPC	72.8	78.1	73.5	76.4	74.6	0.55
Stagefright	64.5	63.2	65.6	62.9	64.2	0.56

We setup a platform in multiple production mobile networks to collect their TCP throughput trace data. The server host ran Linux Apache httpd [26] serving video data over TCP CUBIC [27]. The client was a notebook computer running Microsoft Windows 10 connected to the mobile network via a 3G/LTE USB modem. We developed a custom software to automatically initiate long TCP sessions to measure the actual throughput achievable over the mobile networks.

At the time of writing, a total of 44 weeks' trace data were captured using a stationary client in three locations under two mobile operators. The dataset is publicly available at [28]. Table 1 summarizes the datasets' characteristics, showing considerable variations.

Table 2 summarizes the system settings adopted in the simulations. Through collaboration with an anonymous mobile operator we obtained an empirical video duration distribution (ranges from 40 seconds to 600 seconds) in one of their streaming servers for use as video duration distribution in the simulations. The available video bitrates followed Apple's profile [16] augmented by two more bitrates at 10 Mbps and 12 Mbps. The initial video bitrate during prefetch is set to the lowest bitrate (i.e., 200 kbps) in accordance to the industry norm (to minimize startup delay) and the client begins playback after buffering 10 s video. Upon rebuffering the player will pause playback and resume when a complete video segment is received. The DASH player was configured with a buffer capacity of 60 s. These were consistently applied to all algorithms simulated.

To evalaute streaming performance we adopted the QoE function proposed by Yin *et al.* [5]:

$$Q = \frac{1}{K}\left(\sum_{k=1}^{K} r_k - \lambda \sum_{k=1}^{K-1} \left| r_{k+1} - r_k \right| - \mu Z_p - \mu_\theta Z_\theta \right) \qquad (1)$$

where Z_p is the total rebuffering duration, Z_θ is the startup delay, r_k is the bitrate selected for segment k, K is the total number of segments and the component weights follow the *Balanced* setting [5], i.e., $\lambda = 1$ and $\mu = \mu_\theta = 3000$. We further consider other QoE metrics in Section 6.4.

We simulated three streaming algorithms: RobustMPC [5] (henceforth called 'MPC'), Pensieve [6], and Stagefright [29]. The first two are current state-of-the-art algorithms in the literature and the third is a hybrid bandwidth-buffer-based algorithm implemented in the Android operating system. For MPC and Pensieve, we used the implementation provided by Mao *et al.* [30]; and we implemented Stagefright based on the Android source [29].

For training Pensieve's neural network[1] we employed 30 days' trace data from each of the four datasets for a total of 120 days' trace data and then randomly selected 2,000 streaming sessions as the training dataset. The rest of the trace data (~50,000 streaming sessions) were then applied to evaluate all three algorithms.

3.2 Results and Discussions

Table 3 compares the streaming performance of the three algorithms simulated. We normalized the actual QoE performance by the *offline* optimal computed according to Spiteri *et al.* [7], which serves as an upper bound on the QoE that an omniscient policy with complete and perfect knowledge of future network throughput can achieve.

Normalized QoE (as opposed to actual QoE) enables us to compare the *efficacy* of an algorithm under different network conditions. Specifically, if we compare the performance of an algorithm operating under good versus poor network conditions, then the actual QoE will naturally be lower under poor network condition. This, however, does not tell us the efficacy of the algorithm under the two network conditions. By normalizing against the upper bound we can then compensate for the inherent QoE differences due to differing network conditions so that the algorithms' efficacy can be compared. For brevity we will use QoE and normalized QoE inter-changeably in the rest of the paper.

From Table 3 both Pensieve and MPC outperformed Stagefright in overall QoE performance. Interestingly, Pensieve achieved better QoE than MPC in datasets #1 and #2 but performed worse in datasets #3 and #4. Noting the datasets' throughput in Table 1 it appears that Pensieve is more effective in networks with higher throughput.

To further analyze the results across different network conditions we divided all streaming sessions into 10 throughput levels, with level l=0,1,…,8 collecting sessions with average throughput within (l, l+1] Mbps, plus level 9

[1]Based on the author's suggestion [30], we experimented 5 neural network models with five different initial entropy weights ranging from 1 to 5 respectively, and linearly reduced the values in a gradual fashion until the entropy weight eventually reached 0.1 (after 100,000 iterations). From the trained 5 models, we then selected the best performing model in the validation.

with average throughput ≥ 9Mbps, and then plotted their respective QoE performance in Fig. 1. It clearly shows the reason for the QoE variations in Pensieve – the normalized QoE performance, i.e., efficacy, degraded significantly at the two lowest throughput levels (where dataset #3 and #4 have more of them).

Even more surprisingly, Stagefright achieved significantly higher QoE than Pensieve at throughput level 0. This shows that better-performing algorithm does *exist* for that throughput level, just that it was not adopted by Pensieve during training. We argue that this is due to the fundamental limitation of solo adaptive streaming – the optimal adaptation logics for different network conditions are simply *incompatible*. With only one choice in hand, Pensieve therefore did its best to train an algorithm that maximizes overall QoE performance.

4 Ensemble Adaptive Streaming

In music a solo instrument can only cover a limited range of the scale. To produce the full spectrum of sound one needs an *ensemble* of *different* instruments to cover the full musical scale. Along the same principle we propose a novel Ensemble Adaptive Streaming (EAS) paradigm to incorporate a set of *complementary* adaptation algorithms for which the matching algorithm can then be dynamically selected at *runtime* for use in *each* streaming session. We first investigate in this section the application of EAS to Pensieve and explore a new approach in Section 5.

4.1 Network Differentiators

In EAS the system is equipped with a set of adaptation algorithms. The key is that the algorithms should be *complementary*, i.e., they should each cover (and thus optimized for) a *subset* of the target network environments but together, provide full coverage. Consequently the first challenge is to devise means to segregate network environments into disjoint classes so that specialized adaptation algorithms can be designed/optimized for each one.

To tackle this we introduce the notion of *network differentiators* (ND) $\{V_i, i=0,\ldots,N\text{-}1\}$ which are quantitative metrics that can be obtained, measured, or estimated from the network the system operates in for each streaming session. Each ND, i.e., V_i, computes into a scalar value over a range, $V_{i,min} \le V_i \le V_{i,max}$, which can then be sub-divided into classes $W_i=0,1,\ldots,M_i\text{-}1$, according to a mapping policy Ψ_i:

$$V_i \xrightarrow[\Psi_i]{} W_i \tag{2}$$

Together the chosen NDs then form a ND vector

$$\overrightarrow{W}=\langle W_0, W_1, \ldots, W_{N-1} \rangle \tag{3}$$

which divides the problem space into

$$\Omega=\prod_{i=0}^{N-1} M_i \tag{4}$$

classes for which each class will have its own specialized adaptation algorithm. Obviously the choice of NDs is one of the keys to EAS. We draw on the observations in Section 3 to define a ND in the next section.

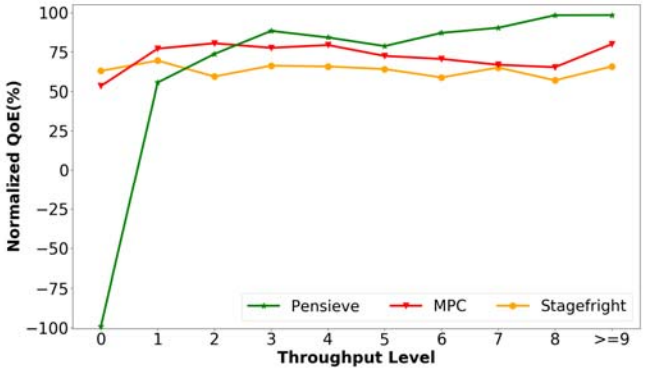


Fig. 1. Comparison of QoE performance across throughput levels.

4.2 Session Throughput Levels

According to the results in Section 3, efficacy of the three algorithms evaluated were impacted substantially by the level of session throughput. This suggests that session throughput could potentially be a good network differentiator. By dividing the problem space using session throughput as the ND we can then design *separate* algorithms for each network class to match its characteristics.

After offline training the ensemble of adaptation algorithms are either loaded into the client in client-driven adaptation or installed into the server in server-driven adaptation. Either way the system will need to select the matching adaptation algorithm, i.e., neural network in the case of Pensieve, for use in *each* new streaming session.

This poses a challenge, however, as while the ND can be calculated during training as the trace data were given, it *cannot* be known at the *beginning* of an actual streaming session. Therefore we need a way to estimate the ND. Moreover, the same estimation method should also be adopted in training despite the availability of full trace data so that the training environment is the same as testing.

The estimation method for a ND depends on its type, its definition, and the environment it operates in. For session throughput we propose two methods to estimate it:

Method 1: **inter-stream estimation**. If a user watches multiple videos in a roll then we could estimate the ND for the new streaming session j from the session throughput of the previous completed streaming session j-1:

$$V_{0,j} = \frac{1}{n}\sum_{k=0}^{n-1} \frac{s_{j\text{-}1,k}}{d_{j\text{-}1,k}} \tag{5}$$

where n is the number of segments, $s_{j\text{-}1,k}$, and $d_{j\text{-}1,k}$ are size and download time for segment k in session j-1.

There are limitations to this method. For example, it may be the first video a user watches or an extended period of time has elapsed since the completion of the previous streaming session so that the network environment may have changed already.

Method 2: intra-stream estimation. To tackle inter-stream estimation's limitation we propose a second method to estimate the ND based on measurement within the *new* streaming session j. In particular, client video player typically prefetches a number of video segments (or video duration) before commencing playback.

Table 4 - Normalized QoE Performance (%) and Coefficient of Variation (CoV) for EAS-Pensieve.

Algorithm	Dataset					
	#1	#2	#3	#4	Overall QoE	Overall CoV
EAS-Pensieve-1	86.8	86.4	78.9	83.1	84.4	0.70
EAS-Pensieve-2	85.7	85.6	76.0	81.3	82.9	0.69
Pensieve	79.2	79.7	56.3	67.0	72.5	0.91
MPC	72.8	78.1	73.5	76.4	74.6	0.55
Stagefright	64.5	63.2	65.6	62.9	64.2	0.56

The throughput in downloading the prefetch segments reflects the current network condition and thus could offer an estimation of the ND. Most importantly this eliminates the dependency on a previously completed streaming session.

Let α be the pre-configured bitrate for the first m segments during prefetch, i.e.,

$$r_{j,k} = \alpha, k = 0,1,\ldots m-1 \tag{6}$$

where $r_{j,k}$ denotes the selected video bitrate for the k^{th} segment in session j. After segment m-1 is received the system can then estimate the ND from

$$V_{0,j} = \frac{1}{m}\sum_{k=0}^{m-1}\frac{s_{j,k}}{d_{j,k}} \tag{7}$$

where $s_{j,k}$, and $d_{j,k}$ are size and download time for segment k in the prefetch phase of session j.

Nevertheless, there are also tradeoffs to this method, e.g., the prefetch duration is typically short (to reduce startup delay) so the number of measurement samples will be small. This may impact the estimation accuracy for certain types of ND (c.f. Appendix A.2).

For mapping policy we employed a linear quantization policy (c.f. Appendix A.2 for other policies):

$$W_0 = \min\left(\left\lfloor\frac{V_0}{\Delta_0}\right\rfloor, M_0 - 1\right) \tag{8}$$

where Δ_0 is the quantization step size and M_0 is the number of classes for ND V_0. With only a single ND the ND vector is simply equal to

$$\overrightarrow{W} = \langle W_0 \rangle \tag{9}$$

The second step in EAS is to design and optimize *separate* adaptation algorithms for *each* network class. In this section we employed the leading machine learning method Pensieve [6] as the tool to design separate neural networks for use in each network class.

Specifically, we modified Pensieve's offline training phase by first dividing all streaming sessions S_j, j=0,1,…,N, into M_0 network classes according to (9):

$$C_i = \left\{S_j \middle| \overrightarrow{W_j} = \langle i \rangle, \forall j\right\}, \; i = 0,1,\ldots,M_0 - 1 \tag{10}$$

where $\overrightarrow{W_j}$ is the ND vector for streaming session j.

Finally we ran separate Pensieve training instances, denoted by the function $T_{Pensieve}(\cdot)$, for each network class to obtain M_0 neural networks, denoted by η_i, for use in each of the matching network classes:

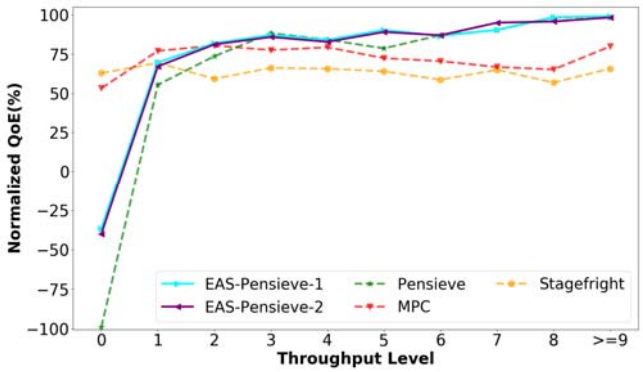


Fig. 2. Comparison of QoE performance across throughput levels.

$$\eta_i = T_{Pensieve}(C_i), \; i = 0,1,\ldots,M_0 - 1 \tag{11}$$

4.3 Performance Evaluation

In the following we analyze the performance of applying EAS to Pensieve using session throughput as the ND. We implemented two versions: EAS-Pensieve-1 and EAS-Pensieve-2 which employed inter- and intra-stream estimations respectively and compare them to the original Pensieve [6], MPC [5], and Stagefright [27]. We adopted the linear mapping policy in (8) with bin size of Δ_0 = 1 Mbps and M_0=10 network classes. We used 30 days' trace data from each of the 4 datasets in Table 1 and randomly selected 200 streaming sessions for training each network class. Other settings are the same as Section 3.

Table 4 summarizes the test subjects' normalized QoE performance and their coefficient of variation (CoV) across the 4 datasets. First, performances of Pensieve in dataset #3 and #4 were improved significantly by EAS, i.e., from 56.3% and 67.0% to 76.0%/78.9% and 81.3%/83.1% respectively. As these two datasets have lower average bandwidth (c.f. Table 1) this suggests that EAS successfully enabled Pensieve to train better-matching neural networks to deal with the more challenging network conditions.

Furthermore, EAS was able to improve Pensieve's performance even for datasets #1 and #2, suggesting that relieved of the burden to cater to lower bandwidth networks, EAS enabled Pensieve to train more effective neural networks to exploit the higher bandwidth available. It is worth noting that the overall performance of original Pensieve was sufficiently impacted by dataset #3 and #4 such that it underperformed MPC in overall QoE performance. By contrast, EAS enabled Pensieve to fully exploit its potential to outperform MPC.

Comparing EAS-Pensieve-1 with inter-stream ND estimation to EAS-Pensieve-2 with intra-stream ND estimation we can quantify the performance tradeoffs in the latter: it ranges from 0.8% to 2.9% across the four datasets with an average of 1.5%. This suggests that intra-stream estimation offers a good alternative for practical deployment.

To further analyze EAS-Pensieve's performance, we plot in Fig. 2 its normalized QoE across the 10 throughput levels. The suboptimal performance of Pensieve at the low throughput extreme observed in Section 3 was improved

substantially by EAS. Unexpectedly, at the lowest throughput level 0, Stagefright still managed to perform significantly better than even EAS-Pensieve. We conjecture that despite Pensieve's use of deep learning approach, its specific neural network topology may not be sufficiently flexible to explore the complete solution space. To this end we turn to a new approach to designing and optimizing adaptation logics – genetic programming.

5 Evolving Adaptation Algorithms via GP

Genetic programming (GP) [14] was inspired by the process of natural selection in nature where a population evolves and adapts itself to the changing environment through crossover, mutation, and reproduction. GP presents two desirable features to the application of EAS to adaptive streaming. First, GP encodes candidate solutions using *expression trees* [14] which are particularly suitable for representing algorithms. In fact, many existing adaptive streaming algorithms can be mapped to relatively simple expression trees. Second, apart from the tree structure GP does not impose a rigid structure on the solution. The evolution process is free to explore the solution space that can be represented by expression trees. This potentially can resolve the limitation of Pensieve at the lower throughput levels.

In this section we apply GP to evolve the ensemble of adaptation algorithms for use in Ensemble Adaptive Streaming. We adopted the same principles in EAS and applied GP to evolve separate adaptation logic for each network class according to the chosen network differentiator. It is worth noting that other evolutionary or machine learning approaches may also be applied in a similar manner and thus could be a fruitful direction for future work.

5.1 Adaptation Logics as Expression Trees

Compared to genetic algorithm [31] where candidate solutions are typically encoded as a bit-string, GP encodes candidate solutions using *expression trees* which comprise two types of components: *operands* and *operators* as illustrated in Fig. 3.

We capture network and system states as inputs in a GP expression tree via *variable operands*. In particular, we define a set $\Re$ of four domain-specific inputs:

$$\Re=\{c,u,b,q\} \tag{12}$$

where c is the average TCP throughput in downloading the past x (e.g., x=5) video segments; u is the current buffer occupancy measured in playback time at the time of requesting the next segment; b is the bitrate of the last video segment; and q is remaining video duration - the duration of video segments which have not yet been downloaded.

In addition to input variables, adaptive streaming algorithms often incorporate numeric constants and multiplying coefficients in computing the video bitrate (e.g., n_1 to n_4 in Fig. 3). To represent these we define a set $\Im$ of GP operands which comprises numeric constants randomly generated over a given range D:

$$\Im=\{x\in\mathbb{R},-D<x<D\} \tag{13}$$

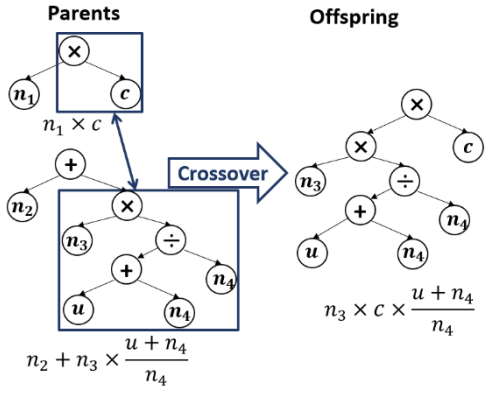


Fig. 3. Illustration of GP expression tree and the process of crossover in reproduction. (c, u are estimated throughput and buffer occupancy respectively; $n_1 \ldots n_4$ are numeric constants).

The non-leaf nodes of an expression tree are operators, which are functions that take inputs to calculate an output value. As depicted in Fig. 3, the inputs themselves can also be a subtree comprising another function and so on. In this work, we employed a set of four arithmetic functions as operators:

$$\aleph=\{+,-,\times,\div\} \tag{14}$$

To execute an expression tree one begins with the root node and then recursively executes each subtree to compute the final numeric output r which is the video bitrate for the next video segment to be downloaded.

In practice there are a finite number of discrete video bitrate choices available so the computed video bitrate r will be mapped to the closest bitrate level available:

$$h=\arg\min_h |\tau_h - r| \tag{15}$$

where τ_h, h=0,1,…,H-1 are the actual bitrate of video bitrate level h for the next video segment.

To illustrate, let us consider the three expression trees in Fig. 3. The top-left tree represents a *bandwidth-based* algorithm which selects future bitrate from the product of the estimated throughput c and the constant n_1. In contrast, the bottom-left tree represents a *buffer-based* algorithm which adapts the bitrate according to the current client buffer occupancy u. Finally, the right-side tree represents a *hybrid-bandwidth-buffer-based* adaptation logic which is the equivalent of the one proposed by Liu *et al.* [8].

It is worth noting that the choices of operands and operators determine the solution space for GP. A desirable property of the above choices in (12) and (14) is that the resultant expression trees can all be mapped directly into *mathematical equations* as illustrated in Fig. 3. This simplifies implementation significantly as incorporation of a GP-evolved solution is as simple as replacing the equation adopted in the adaptation logic.

5.2 The Evolutionary Process

In GP, a population of candidate solutions to an optimization problem is evolved towards better solutions through an evolutionary process [14]. In the following, we present the methodology adopted for evolving adaptive streaming algorithms.

Initialization. The evolutionary process begins with an initial population of κ (e.g., κ=800) randomly-generated individuals. We adopted the method proposed by Koza [14] in generating the initial population with a maximum tree depth of 5.

Fitness evaluation. Each generation of population evolves by means of producing offspring to form the next generation. This is done by first evaluating the *fitness* of each individual in the population which indicates the goodness of each solution in the problem domain. This presents a challenge as the performance of an adaptive streaming algorithm not only depends on the adaptation logic, but also depends on the network conditions as well as the QoE metric adopted.

To tackle this challenge, we propose to employ trace-driven simulation to evaluate the actual performance of a given individual according to a given QoE metric over a period of time. To ensure the fitness evaluation covers a broad range of network conditions each individual's fitness is evaluated over L (e.g., L=200) streaming sessions randomly selected from all 4 datasets in Section 3.

Let $I_k \in \mathbf{I}$ as the k^{th} individual out of the set of population $\mathbf{I}$. Now given the throughput trace data of session j in dataset $\mathbf{E}$, denoted by $S_j \in \mathbf{E}$, expression tree I_k can be executed (denoted by the function $F(\cdot)$) to produce a set of performance metrics (e.g., average bitrate, rebuffering duration, etc.), collectively denoted by $P_{k,j}$:

$$P_{k,j} = F(I_k, S_j) \tag{16}$$

Finally, the fitness of the k^{th} individual, denoted by f_k, is computed from the mean QoE of all L streaming sessions:

$$f_k = \frac{1}{L}\sum_{j=0}^{L-1} U(P_{k,j}) \tag{17}$$

where $U(\cdot)$ is the QoE function adopted.

Selection. Once the fitness for all individuals in the population is obtained, GP then performs *selection* (e.g., tournament selection [32]) among individuals to generate offspring for the next generation. Individuals with better fitness will stochastically have more chance to reproduce, i.e., the process of selection is guided by the fitness metric (i.e., QoE). Selection gives preference to better individuals to allow them to pass on their genes to the next generation.

Crossover and mutation. After selection, GP randomly explores the combination of good genes in selected individuals through *crossover* – a process where a sub-tree from each of two parent individuals are swapped to form an offspring. This is illustrated in Fig. 3 where the two parent individuals on the left perform crossover to form an offspring that integrates both bandwidth information c and buffer information u into its adaptation logic. In addition to crossover, GP also implements *mutation* where a sub-tree in an individual is replaced by another randomly-generated sub-tree using the same random tree generation method as used in the initial population. Mutation increases the diversity of the population to broaden the search space and to enable GP to escape local minima.

Termination. The set of reproduced offspring then forms the next generation and the process repeats until it reaches a pre-defined maximum number of generations G (e.g., G=50). Generation after generation, GP can then explore a wide spectrum of candidate solutions in the solution space to progressively evolve better adaptation algorithms. The best individual from the final population will then be selected as the adaptive streaming algorithm for use in actual streaming sessions.

5.3 System Architecture

EAS-GP operates in two phases: an offline evolutionary phase and an online streaming phase. The offline phase runs separately from the streaming platform except that it makes use of throughput trace data collected from the latter. This can be done by capturing (e.g., tcpdump) the throughput trace data at the servers as a byproduct of actual streaming sessions or via measurements as described in Section 3.

To evolve *separate* adaptation algorithms for *each* network class, we first divide all training streaming sessions S_j, j=0,1,…,N into M_0 network classes according to (9) and (10). For each network class i we run a separate GP evolution process, denoted by the function $T_{GP}(\cdot)$, using trace data subset C_i for fitness evaluation as described in Section 5.2. This enables GP to evolve specialized algorithms for each network class.

After the evolution process is terminated the expression tree with the highest fitness value, denoted by ε_i, will be adopted for use in each of the matching network classes:

$$\varepsilon_i = T_{GP}(C_i), \quad i = 0,1,\ldots,M_0 - 1 \tag{18}$$

Finally we transform the M_0 expression trees into their mathematical equation counterparts for use in either the server (server-driven adaptation) or the client (client-driven adaptation). In either case, the online streaming platform itself does *not* need any GP components. The only modifications needed are: (a) an extra module to estimate the network differentiators during the prefetch phase; (b) an initial step to select using the estimated NDs the matching equation for bitrate adaptation according to the network class the streaming session is classfied into. The rest of the streaming process will be identical to ordinary adaptive streaming so implementation of EAS-GP can be readily be applied to existing or new streaming platforms.

To demonstrate EAS-GP's practicality we implemented a version of it into the well-known dash.js video player [33] by simply modifying the "changeQuality" function in the "AbrController" class [34] with 154 lines of codes. The trained expression trees are represented as text (in a prefix-notation) that can either be downloaded separately to the client or even embedded within the video playlist. The trained expression trees are typically small in size (e.g., ~9KB in our experiments) so the extra overhead is negligible. A demo of EAS-GP implemented via dash.js can be tested online at [35].

6 Results and Discussions

In this section we evaluate performance of the EAS-optimized algorithms and compare them to current state-of-the-art adaptive streaming algorithms.

Table 5 - Comparison of Normalized QoE Performance (%) and Coefficient of Variation (CoV).

Algorithm	Dataset #1	#2	#3	#4	Overall QoE	Overall CoV
EAS-GP-1	90.2	90.6	87.2	90.1	89.6	0.60
EAS-GP-2	87.1	87.4	84.8	87.0	87.0	0.60
Solo-GP	84.6	85.3	78.7	84.4	84.4	0.62
EAS-Pensieve-1	86.8	86.4	78.9	83.1	84.4	0.70
EAS-Pensieve-2	85.7	85.6	76.0	81.3	82.9	0.69
Pensieve	79.2	79.7	56.3	67.0	72.5	0.91
MPC	72.8	78.1	73.5	76.4	74.6	0.55
Stagefright	64.5	63.2	65.6	62.9	64.2	0.56

6.1 Methodology

We employed trace-driven simulation as described in Section 3 for performance evaluation. The system settings are the same as described in Section 3. Sensitivity analysis of key system parameters can be found in Appendix A.2. A total of 8 algorithms were simulated: Android's Stagefright [29], Robust MPC [5], original Pensieve [6], EAS-Pensieve-1 (with inter-stream ND estimation), EAS-Pensieve-2 (with intra-stream ND estimation), Solo-GP (i.e., evolving a single algorithm only), EAS-GP-1 (with inter-stream ND estimation), and EAS-GP-2 (with intra-stream ND estimation).

Unless stated otherwise the training dataset is composed of 30 days' trace data from each of datasets #1 to #4. We employed estimated session throughput as the network differentiator using the linear quantization policy in (8) to divide session into M_0=10 network classes with quantization step size of Δ_0=1 Mbps. In the GP evolution process we adopted a population size of 800; fitness was evaluated using the same set of training trace data as used in Pensieve; the population was evolved for 50 generations after which the expression tree with the best fitness (i.e., highest QoE according to (17)) was then adopted for online performance evaluation using unseen trace data.

The performance metric is normalized QoE as described in Section 3. Note that the QoE function in (1) can be configured via component weights into three versions: *Balanced*, *Avoid Rebuffering*, and *Avoid Instability* [5]. Except for Stagefright which is QoE function agnostic, the same QoE function was used in both training and testing for MPC, Pensieve, and GP unless stated otherwise.

6.2 QoE Performance Comparisons

Table 5 compares the normalized QoE of all algorithms tested. First, Solo-GP performed remarkably well against existing algorithms, matching the performance of EAS-Pensieve-1. Moreover, EAS further improved GP's performance by 2.5% to 10.8% across the four datasets, reaching close to or over 90% normalized QoE performance. Given that 100% normalized QoE is *not* realizable (requires complete trace data *a priori*), the level of performance achieved by EAS-GP is remarkably close to the upper bound.

Second, comparing EAS-GP-1 with EAS-GP-2 shows that inter-stream ND estimation can achieve slightly higher (2.4% to 3.2%) performance than intra-stream ND estimation, consistent with the findings for EAS-Pensieve-1 and EAS-Pensieve-2.

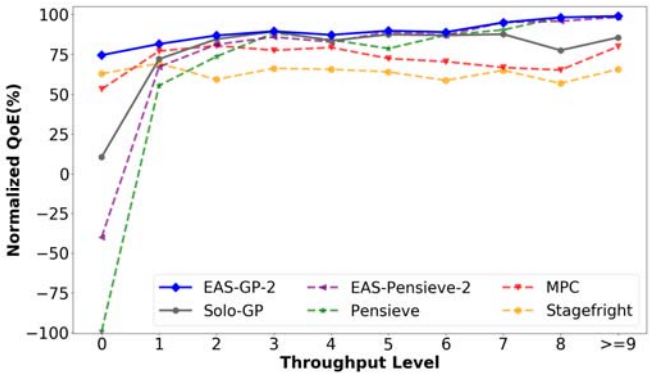


Fig. 4. Comparison of QoE performance across throughput levels.

To analyze the algorithms' performance across different network conditions we plot in Fig. 4 the QoE versus throughput levels from 0 to 9. Since the performance of inter-stream ND and intra-stream ND are very close, we plot only the intra-stream ND curves for EAS-Pensieve and EAS-GP to reduce clutter.

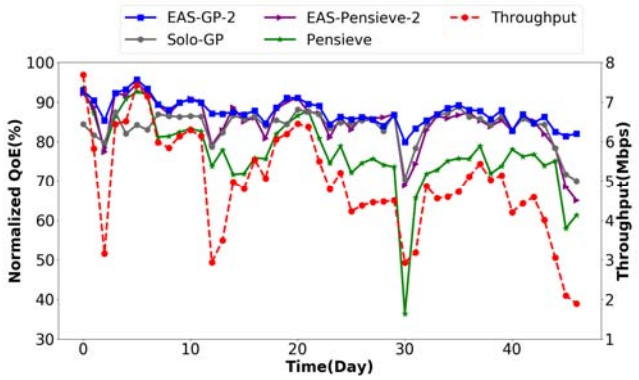
We observe that although Solo-GP performed well in overall QoE, its performance at the lowest throughput level 0 is still substantially lower than Stagefright. This is a fundamental limitation to any solo-algorithm approach as it is difficult, if not impossible, for a single adaptation logic to work well for all kinds of network conditions. Unlike Pensieve however, the more flexible expression tree structure of GP enabled EAS to evolve specialized algorithms for all 10 throughput levels. In fact, at throughput level 0 EAS-GP outperformed Stagefright. Similarly, comparing to Solo-GP, EAS-GP also achieved substantially higher QoE at the high throughput levels.

The above results clearly demonstrate the fundamental limitation of solo adaptive streaming. Moreover, merely applying EAS, despite the substantial performance gains in EAS-Pensieve, may not be sufficient on its own. One also needs a platform for which EAS can fully explore and exploit specializations to match the wide range of network conditions for optimal performance. The GP platform presented in Section 5 offers one such platform which worked remarkable well. More work is warranted to explore other machine learning or evolutionary computation platforms to see if one can push the envelope even further.

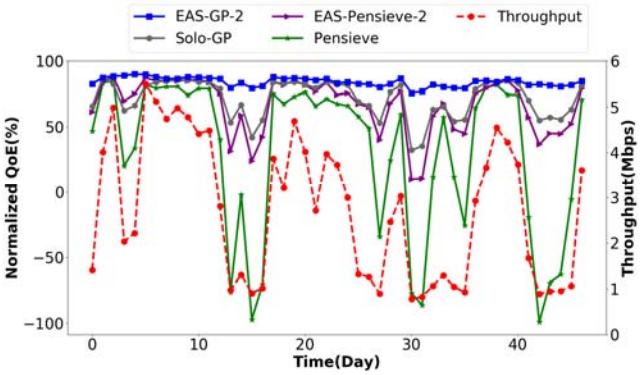
6.3 Robustness Analysis

An often neglected dimension of algorithms trained and optimized from trace data is their robustness. Specifically, we consider two robustness metrics in this section, namely *temporal robustness* and *spatial robustness*.

Temporal robustness concerns an algorithm's performance consistency over time. Fig. 5 plots the algorithms' QoE performance for dataset #1 and #3 over time where the x-axis is the number of days after training was completed. We also plot daily-averaged throughput in the same graph to show how the average throughput varied from day to day.



(a) Dataset #1



(b) Dataset #3

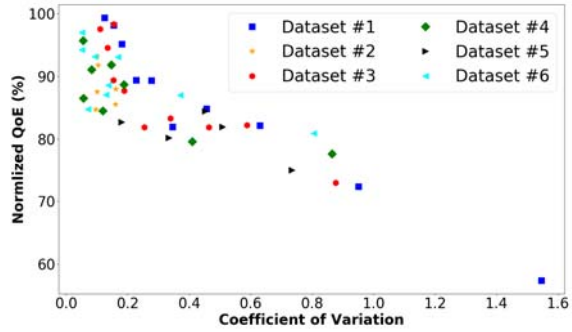
Fig. 5. Comparison of temporal robustness.

Table 6 - Comparison of Spatial Robustness via Normalized QoE (%).

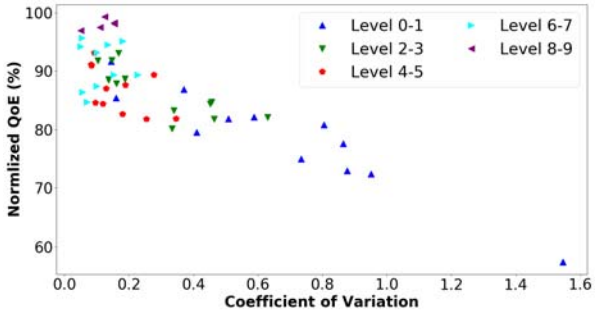
Algorithm	Dataset					
	#1	#2	#3	#4	#5	#6
EAS-GP-2	87.3	87.3	84.4	86.6	76.5	84.1
Solo-GP	84.3	85.1	75.4	83.2	55.2	79.4
EAS-Pensieve-2	84.2	84.7	69.9	78.8	37.0	76.4
Pensieve	81.6	79.5	51.5	69.0	21.7	70.1

The first observation is that daily averaged throughput can vary quite significantly even though the trace data were captured in the *same* physical location by a *stationary* client. This will pose challenges to solo adaptive streaming and the results indeed confirmed that – original Pensieve and to a lesser extent, Solo-GP, both exhibited substantially more variations than their EAS counterparts. By contrast, EAS-GP-2 consistently achieved the highest QoE performance across *all* 47 days tested. This suggests that with EAS-GP-2 one may *not* need to *re-evolve* the algorithm at all as the variations are largely due to network condition changes which have already been addressed by EAS.

Spatial robustness concerns an algorithm's performance consistency over different network characteristics, e.g., geographical locations and service providers, which presumably may exhibit different ranges of network conditions. Previously, GP and Pensieve were trained using trace data from all 4 datasets. An interesting scenario is to train using just *one* trace dataset and then test using *different* datasets.



(a) Color by dataset.



(b) Color by throughput level.

Fig. 6. Normalized QoE of EAS-GP-2 versus throughput coefficient-of-variation (CoV). Each marker represents one throughput level from a dataset.

Table 6 summarizes the results of such an experiment where an algorithm is trained using 60 days' trace data from dataset #1 and then tested in dataset #1 to #6 (in each case all unseen trace data in a dataset were used for testing). We note that datasets #1 to #4 were captured by ourselves in production mobile networks with a stationary client while dataset #5 (mobile network with client mobility) and #6 (wired network) were obtained from [36] and [37] respectively.

Not surprisingly, the algorithms' QoE performances varied across testing datasets. Pensieve and to a lesser extent EAS-Pensieve-2 did exhibit larger variations, e.g., with QoE dropping to 21.7% and 37.0% in dataset #5 (which has a mean throughput of only 1.19 Mbps). This is due to Pensieve's tendency for more aggressive bitrate selections (c.f. Section 6.6 and Appendix A.1). In comparison, Solo-GP is more robust, e.g., maintaining a QoE of 55.2% in dataset #5. EAS-GP-2 achieved the highest spatial robustness, maintaining a QoE of 76.5% even in the challenging dataset #5, and QoE of over 84% in all other datasets.

To further analyze factors that impact QoE performance we plot in Fig. 6(a) the normalized QoE of EAS-GP-2 versus throughput coefficient-of-variation (CoV) for each throughput level from the 6 datasets. We observe a clear relation between throughput CoV and QoE which is consistent with intuition as the more variable the throughput is, the more difficult it is to adapt the video bitrate. An interesting observation is that high CoV is not limited to dataset #5 and the latter also has throughput levels with low CoV.

Table 7 - Proportion of Video Sessions (%) Across Throughput Levels

Dataset	Throughput level				
	0-1	2-3	4-5	6-7	8-9
#1	6.5	18.8	47.1	24.2	3.4
#2	14.2	43.6	38.1	4.0	0.1
#3	52.7	18.3	18.3	9.3	1.4
#4	28.0	46.7	23.8	1.5	0
#5	85.6	13.3	1.1	0	0
#6	27.5	44.9	23.2	3.3	1.1

Table 8 - Comparison of Predicted QoE and Actual QoE of EAS-GP-2

Dataset	Predicted QoE	Actual QoE
#2	2747	2918
#3	2015	2065
#4	2037	2277
#5	720	793
#6	2101	2392

If we replot the same figure by assigning colors according to throughput levels in Fig. 6(b) then it becomes clear that QoE/CoV are also correlated to throughput level itself. Generally speaking the lower the throughput level, the higher the CoV and the lower the QoE. Again this is consistent with intuition as throughput is directly affected by the network condition. This also explains why session throughput works well as a network differentiator.

Given the correlation between session throughput CoV and QoE one may wonder if the former could be a good network differentiator as well. We did test this idea and found that it did not offer any significant performance gains as estimating session throughput CoV, especially using intra-stream estimation, is highly error-prone, thereby resulting in frequent misclassifications. More work is thus warranted to investigate this topic further.

Finally, to understand why EAS-GP-2's QoE performance is lower in dataset #5 we turn to Table 7 which summarizes the proportion of throughput levels in each dataset. Note in dataset #5 85.6% of the sessions belong to throughput level 0 and 1. As these low throughput levels generally have lower QoE therefore the overall QoE becomes lower in dataset #5.

A deeper analysis of spatial robustness requires differentiation between an algorithm's *inherent efficacy* versus its effectiveness under the given network condition (i.e., throughput level). To illustrate the idea let us reconsider EAS-GP-2's performance across throughput levels in Fig. 4. It is clear that its QoE increased with higher throughput levels as discussed earlier. The set of QoE for each throughput level thus represents the algorithm's inherent efficacy.

If an algorithm is spatially robust then we would expect it to exhibit similar effectiveness even in completely different locations. To test this idea, let Q_i be the mean *absolute* QoE for throughput level i for an algorithm trained and tested using dataset #1. Different datasets will have different compositions of throughput levels, e.g., dataset #5 has mostly lower throughput levels, and to account for that we can apply weighted average to calculate the *predicted* absolute QoE performance in the target dataset from

Table 9 - Comparison of Normalized QoE Performance (%) against QoE metrics. Except for EAS-GP-3 (see text) all others were trained using the Balanced QoE metric.

Algorithm	QoE Metrics				
	Balanced	Avoid Rebuffering	Avoid Variation	QoE-log	QoE-HD
EAS-GP-3	85.4	84.1	86.2	89.4	82.7
EAS-GP-2	87.0	68.1	86.4	58.8	87.6
Solo-GP	84.4	83.1	83.5	81.3	83.6
EAS-Pensieve-2	82.9	64.2	83.3	55.3	88.5
Pensieve	72.5	13.5	71.8	18.7	90.8
MPC	74.6	74.7	56.5	81.7	72.3

$$\xi = \sum_{i=0}^{M_0-1} Q_i \gamma_i \tag{19}$$

where γ_i is the proportion of video sessions in throughput level i in the target dataset and M_0 is the total number of throughput levels. The physical meaning of (19) is what the absolute QoE performance would have been if the algorithm's efficacy is the same in the target dataset as in dataset #1.

The results for EAS-GP-2 are summarized in Table 8 for target datasets #2 to #6. It is clear from the results that the actual absolute QoE achieved is similar to the predicted one, suggesting that EAS-GP-2 performed consistently even when trained in one dataset and then tested in a completely different dataset. Differences in normalized QoE performance is largely due to different throughput level compositions.

The above results point to an important characteristic of EAS-GP – it is both temporally and spatially robust. This strongly suggests that although EAS-GP was trained using trace data, its ensemble of algorithms is sufficiently general that it could be applied to a much wider range of networks covered by the network classes. Moreover, it may not even be necessary to retrain it periodically as variations in daily throughput has already been accounted for by EAS.

6.4 Revisiting QoE Metrics

Another interesting question is on the choice of QoE metric. In the literature many different QoE metrics have been proposed and obviously an algorithm's performance will depend on the QoE metric chosen. No matter the choice, an algorithm is always tested using the same QoE metric as used in training.

In practice, one can envision that different users may have different QoE preferences (some may prefer higher video quality while others prefer less rebuffering). The problem is that these preferences are currently not known and thus one can only train the adaptation algorithm using one QoE metric.

To investigate this problem we first conducted an experiment to train the algorithms using the Balanced version [5] of the QoE function in (1) and then tested it in 5 different QoE functions/variations, namely, Balanced, Avoid Rebuffering [5], Avoid Variations [5], QoE-log [6],

and QoE-HD [6]. The QoE-log function applies log to the video bitrate to reflect diminishing return as the bitrate increases. In contrast, QoE-HD favors high video quality by giving more weights to the higher bitrate choices. The results are summarized in Table 9.

As expected, the choice of QoE function in testing affects the algorithms' performance. MPC performed relatively consistently except for Avoid Variations, suggesting that MPC switches bitrates more often than others. Pensieve's QoE degraded significantly under Avoid Rebuffering and QoE-log functions. This is due to Pensieve's inherently more aggressive bitrate choices (c.f. Section 6.6) which led to higher video bitrate (discounted under both QoE functions) at the expense of more rebuffering. By contrast, it performed best under QoE-HD where its more aggressive bitrate choices were rewarded. This behavior carried over to EAS-Pensieve-2 albeit in a far less extent.

Solo-GP is surprisingly robust to QoE functions while EAS-GP-2 degraded under Avoid Rebuffering and QoE-log. This suggests that EAS allows better optimization (87.0% vs 84.4% in Solo-GP) when the testing QoE is same as in training but becomes over-specialized under dissimilar QoE functions.

In principle, to avoid over-specialization one can expose the GP population to a more diverse environment. To explore this idea we ran another experiment where the fitness evaluation step was modified to randomly select one out of the five QoE functions for each video session so that the population will be exposed to all five QoE functions. The results are shown in Table 9 in the row named EAS-GP-3. Compared to EAS-GP-2 the degradations in Avoid Rebuffering and QoE-log have been eliminated. In fact EAS-GP-3 now performed best under these two QoE functions. The tradeoffs are modest decrease in performance under the other three QoE functions. Overall, the gain in robustness far outweighs the slight losses elsewhere, suggesting that employing multiple QoE functions (with sufficiently diverse preferences) during training could be a key to solving the QoE-diversity challenge.

Another direction for future research is to explore ways to capture user's QoE preference as another network differentiator, e.g., offering a slider in the user interface to enable the user to control the tradeoff between video quality and rebuffering. In this way EAS-GP can then be used to evolve specialized algorithms for different QoE preferences, potentially achieving even better performance without the tradeoffs.

6.5 Revisiting Segment Size

All the experiments so far adopted a fixed video segment duration of 2 seconds. Intuitively the choice of segment duration should be the same in both training and testing as it directly affects the adaptation interval as well as measurement of the network environment.

Normally this would not present a problem as long as the service provider can control the segment size. Otherwise it is also straightforward to incorporate segment size as an additional network differentiator to evolve specialized algorithms for it.

Table 10 - Impact of Segment Size on Normalized QoE (%). EAS-GP-4 incorporates Segment Size as an extra ND.

Algorithm	Segment Size for Training (s)	Segment Size for Testing (s)		
		2	4	10
EAS-GP-2	2	87.0	62.3	-40.3
EAS-GP-2	4	81.9	87.7	67.7
EAS-GP-2	10	73.5	82.2	87.3
EAS-GP-4	Determined by ND	87.0	87.7	87.3

Table 10 illustrates this idea by comparing EAS-GP-2's QoE under different pairs of {training, testing} segment sizes. It is clear that the evolved algorithms are sensitive to segment size and this can be easily addressed by incorporating the latter as an extra ND in EAS-GP (denoted by EAS-GP-4).

6.6 Adaptation Logic Analysis

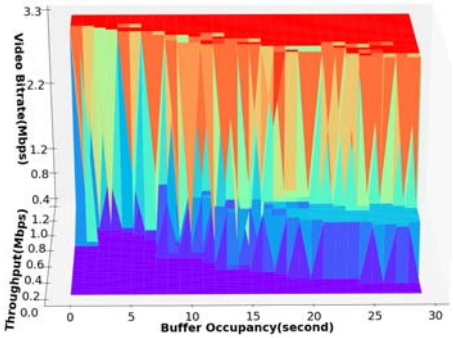
One of the challenges in machine-learning approach to problem solving is that the resultant solutions are often opaque and difficult to analyze so that insights into their performance cannot be easily obtained. In this section we attempt to shed lights on this challenge by presenting a rate-throughput-buffer (RTB) visual approach to analyze the behavior of bitrate adaptation logics.

Specifically, by fixing other less critical parameters, i.e., the last segment bitrate = 200kbps and remaining video duration = 200 seconds, we can plot the bitrate decision (z-axis) versus measured throughput (y-axis) and buffer occupancy (x-axis) as a 3D surface plot for each adaptation algorithm.

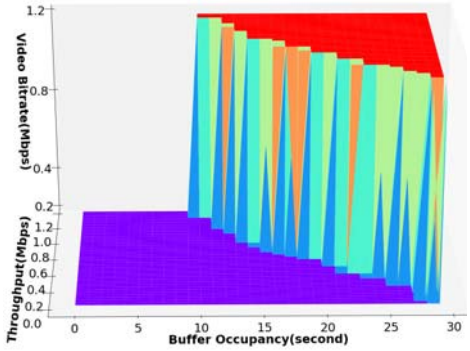
First, we analyze the algorithms' behavior at the lowest throughput level 0 using RTB plots in Fig. 7. We first investigate the original Pensieve where its performance suffered at throughput level 0. Its RTB plot in Fig. 7a revealed one behavior of Pensieve – its bitrate selection logic is relatively aggressive even when the throughput is low. For example, at a measured throughput of around 1 Mbps, Pensieve still selects a bitrate of 3.3 Mbps even if the buffer occupancy is 0. While this may work well in high throughput levels (more on that later) it clearly explains its performance at the lower throughput levels.

Next we investigate EAS-Pensieve in Fig. 7b. Section 4.3 showed that EAS-Pensieve improved Pensieve's performance at throughput level 0 but was still far from optimal. This is confirmed by its RTB plot which shows reduced aggressiveness at lower buffer occupancies - a higher bitrate will not be selected unless both throughput and buffer occupancy reached a certain level. Its bitrate decision boundary, however, is *abrupt* – it changes sharply from 0.2 Mbps to 1.2 Mbps, despite the availability of intermediate bitrate choices (0.4 Mbps and 0.8 Mbps).

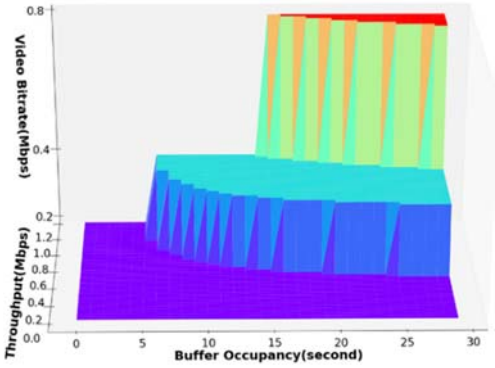
Finally, we investigate EAS-GP's behavior in Fig. 7c. Its RTB plot shows a surprisingly intuitive bitrate adaptation logic – bitrate is gradually increased for higher throughput and buffer occupancy. Moreover, the RTB plot also reveals another important ingredient to the algorithm – its bitrate selection is more conservative than normal. For example, for measured throughput of 1.2 Mbps couple with buffer occupancy of 30 seconds, it will select a video bitrate of only 0.8 Mbps.



(a) Pensieve



(b) EAS-Pensieve

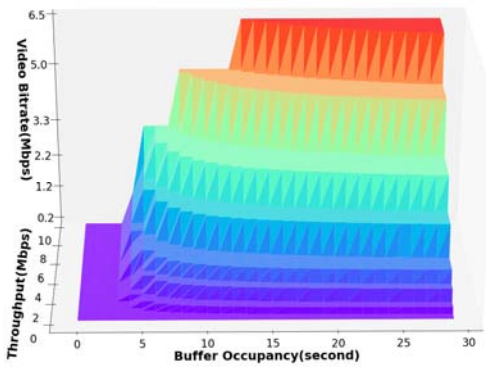


(c) EAS-GP

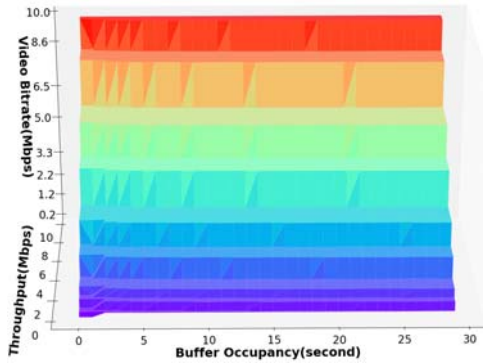
Fig. 7. Rate-Throughput-Buffer (RTB) plots for throughput level 0.

Knowing that it operates at throughput level 0 environment (0~1Mbps), the higher observed throughput is in fact treated by EAS-GP as *exceptions* that is unlikely to last. Therefore not raising the bitrate too far effectively prevents rebuffering in the future.

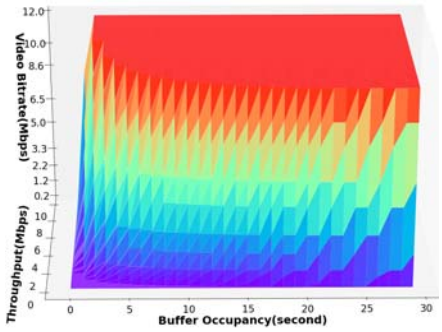
This last point is further illustrated in Fig. 8 which plots the RTB for EAS-GP for low (0), medium (5), and high (9) throughput levels. At level 0 it is clear that EAS-GP intentionally selects bitrates lower than the observed throughput as the network condition is assumed to be poor. By contrast, at level 5 EAS-GP becomes more balanced in its bitrate selection. Finally, at level 9 EAS-GP now becomes more aggressive, often selecting bitrates higher than the observed throughput. Intuitively, at throughput level 9, the lower observed throughput is likely short-term so maintaining high video bitrate can prevent unnecessary QoE degradations.



(a) Throughput level 0 algorithm



(b) Throughput level 5 algorithm



(c) Throughput level 9 algorithm

Fig. 8. Comparison of EAS-GP RTB in throughput levels 0, 5, and 9.

This analysis also explains why one algorithm cannot work well for all network conditions as the desired behaviors under different network conditions are simply *incompatible*.

7 Summary and Future Work

This work uncovered three new insights in the design of adaptive video streaming: (a) it is both possible and practical to quantitatively (via network differentiators) classify networks into different classes and then apply specialized adaptation algorithm to each network class to obtain significant performance gains; (b) machine learning approaches may themselves impose constraints on the representable solutions and as such, a broader exploration and investigation is warranted on the automatic design of adaptation logics; and (c) the proposed EAS-GP approach not only evolved adaptation algorithms that outperformed the

current state-of-the-art, but also exhibited remarkably high spatial, temporal, and even QoE robustness.

The last point is of particular importance as we conjecture that EAS couple with proper choice of network differentiators could offer a way to design adaptation algorithms that can perform consistently well in a very broad range of network conditions and over a long time horizon without the need to retrain for specific locations and time.

Looking forward more work is warranted to explore the choice and design of other network differentiators, to explore the use of other machine learning or automatic design paradigms to train adaptation logics, and to explore EAS's robustness in a wider range of network types (e.g., WiFi, 5G). In addition, this work only considered the single-client case. Nowadays it is not uncommon to have multiple streaming players sharing an Internet connection, e.g., in a home network. Previous works [39-40] have shown that adaptation algorithms may behave unpredictably when sharing a bottleneck as the inherent *on-off* nature of segment downloads could negatively impact throughput estimation. Our initial results have shown that this could be mitigated by allowing unlimited client buffer as that enables the clients to download segments nonstop until completion, thereby allowing TCP to achieve fair bandwidth sharing. However this may not be desirable in some use cases, e.g., due to storage constraint or data wastage [41], and thus more work is warranted to further tackle this challenge.

ACKNOWLEDGEMENTS

The authors wish to thank the associate editor and the anonymous reviewers for their insightful comments in improving this paper. This work was funded in part by a research grant from the HKSAR Research Grant Council and a Direct Grant from CUHK.

REFERENCES

[1] E. Dahlman, S. Parkvall and J. Skold, "4G: LTE/LTE-advanced for Mobile Broadband," *Academic Press*, 2013.

[2] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong and J. C. Zhang, "What will 5G be?" *IEEE Journal on Selected Areas in Commun.*, vol.32(6), Jun 2014, pp.1065-1082.

[3] *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016-2021*, Mar 2017, Cisco Inc. [Online] http://www.cisco.com/c/en/us/so-lutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

[4] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP: Standards and Design Principles," *Proc. ACM Multimedia Syst.*, San Jose, USA, Feb 2011, pp.133-144.

[5] X. Yin, A. Jindal, V. Sekar and B. Sinopoli, "A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP," *Proc. ACM SIGCOMM*, London, United Kingdom, Aug 2015, pp.325-338.

[6] H. Mao, R. Netravali, M. Alizadeh, "Neural Adaptive Video Streaming with Pensieve," *Proc. ACM SIGCOMM*, Los Angeles, CA, USA, Aug 2017, pp. 197-210.

[7] K. Spiteri, R. Urgaonkar and R.K. Sitaraman, "BOLA: Near-Optimal Bitrate Adaptation for Online Videos," *Proc. IEEE INFOCOM*, San Francisco, CA, USA, Apr 2016, pp.1-9.

[8] Y. Liu and J.Y.B. Lee, "Post-Streaming Rate Analysis - A New Approach to Mobile Video Streaming with Predictable Performance," *IEEE Transactions on Mobile Computing*, vol.16(12), Dec 2017, pp.3488-3501.

[9] A. H. Zahran, D. Raca, C. Sreenan. "ARBITER+: Adaptive Rate-Based InTElligent HTTP StReaming Algorithm for Mobile Networks". *IEEE Transactions on Mobile Computing*, vol.17(12), Apr 2018, pp.2716-2728.

[10] Y. Qin, R. Jin, S. Hao, K. R. Pattipati, F. Qian, S. Sen, B. Wang and C. Yue, "A Control Theoretic Approach to ABR Video Streaming: A Fresh Look at PID-based Rate Adaptation," *Proc. IEEE INFOCOM*, Atlanta, GA, USA, May 2017, pp.1-9.

[11] F. Chiariotti, S. D'Aronco, L. Toni, "Online Learning Adaptation Strategy for DASH Clients," *Proc. ACM Multimedia Syst.*, Klagenfurt, Austria, May 2016, pp.8:1-8:12.

[12] V. Martín, J. Cabrera, N. García, "Design, Optimization and Wvaluation of a Q-Learning HTTP Adaptive Streaming Client," *IEEE Transactions on Consumer Electronics*, vol. 62(4), Nov 2016, pp. 380-388.

[13] S. Petrangeli, M. Claeys, S. Latré, J. Famaey, and F. De Turck, "A Multi-Agent Q-Learning-based Framework for Achieving Fairness in HTTP Adaptive Streaming," *IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–9.

[14] J. R. Koza, "Genetic Programming as A Means for Programming Computers by Natural Selection," *Springer Statistics and computing*, vol. 4(2), Jun 1994, pp.87-112.

[15] *Microsoft Inc. Microsoft Smooth Streaming*. [Online] http://www.iis.net/downloads/microsoft/smooth-streaming

[16] *Best Practices for Creating and Deploying HTTP Live Streaming Media for the iPhone and iPad*, Apple Inc, retrieved on Aug 2016. [Online] https://developer.apple.com/library/ios/technotes/tn2224/_index.html

[17] *Adobe HTTP Dynamic Streaming*. [Online] http://www.adobe.com/products/hds-dynamic-streaming.html

[18] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hobfeld and P. Tran-Gia,"A Survey on Quality of Experience of HTTP Adaptive Streaming," *IEEE Commun. Surveys Tuts.*, vol.17(1), Q1 2015, pp.469-492.

[19] P. Juluri, V. Tamarapalli, and D. Medhi, "Measurement of Quality of Experience of Video-on-Demand Services: A Survey," *IEEE Commun. Surveys Tuts.*, vol.18(1), Feb 2016, pp.401-418.

[20] J. Kua, G. Armitage, and P. Branch, "A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming over HTTP," *IEEE Commun. Surveys Tuts.*, vol.19(3), Mar 2017, pp.1842-1866.

[21] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, R. Zimmermann, "A Survey on Bitrate Adaptation Schemes for Streaming Media over HTTP," *IEEE Commun. Surveys Tuts.*, Aug 2018.

[22] Y. Sun, X. Yin, J. Jiang, S. Vyas, "CS2P: Improving Video Bitrate Selection and Adaptation with Data-driven Throughput Prediction," *Proc. ACM SIGCOMM*, Florianopolis, Brazil, Aug 2016, pp. 272-285.

[23] C. J. C. H. Watkins, P. Dayan, "Q-learning," *Machine learning*, vol.8(3-4), May 1992, pp. 279-292.

[24] M. Kearns and S. Singh, "Finite-sample Convergence Rates for Q-learning and Indirect Algorithms," *Advances in Neural Information Processing Systems*, vol.11, 1999, pp. 996–1002.

[25] V. Mnih, A. P. Badia, M. Mirza, A. Graves, "Asynchronous Methods for Deep Reinforcement Learning," *International conference on machine learning*, New York City, NY, USA, Jun 2016, pp. 1928-1937.
[26] *Apache HTTP Server Project.* [Online] http://httpd.apache.org/
[27] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM Operating Sys's Review*, vol.42(5), 2008, pp.64-74.
[28] *Mobile Throughput Trace Data.* [Online] http://sonar.mclab.info/tracedata/TCP/3G/
[29] *The Android Open Source Project. (2015, Feb.)* Android Git Repositories. [Online] https://android.googlesource.com/platform/frameworks/av/+/master/media/libstagefright/httplive/LiveSession.cpp
[30] *Pensieve Source Code* [Online] https://github.com/hongzimao/pensieve
[31] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, MIT press, 1992.
[32] B. L. Miller, D. E. Goldberg, "Genetic Algorithms, Tournament Selection, and the Effects of Noise," *Complex systems*, vol.9(3), 1995, pp. 193-212.
[33] *dash.js*, [Online] https://github.com/Dash-Industry-Forum/dash.js/wiki.
[34] *The Implementation of EAS-GP in dash.js.* [Online] http://dash.mclab.org:3000/src/streaming/controllers/AbrController.js
[35] *The Demo Video Player of EAS-GP.* [Online] http://dash.mclab.org:3000/samples/dash-demo/index.html
[36] H. Riiser, P. Vigmostad, C. Griwodz, P. Halvorsen, "Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications," *Proc. ACM Multimedia Syst.*, Oslo, Norway, Feb 2013, pp.114-118.
[37] Federal Communications Commission, *Raw Data Measuring Broadband America*. [Online] https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016.
[38] Z. Akhtar, Y. S. Nam, R. Govindan, "Oboe: Auto-tuning Video ABR Algorithms to Network Conditions" *Proc. ACM SIGCOMM*, Budapest, Hungary, Aug 2018, pp.44-58.
[39] S. Akhshabi, L. Anantakrishnan, A. C. Begen, C. Dovrolis, "What Happens When HTTP Adaptive Streaming Players Compete for Bandwidth?" *Proc. ACM NOSSDAV*, Toronto, Ontario, Canada, Jun 2012, pp. 9-14.
[40] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, A. C. Begen, "Server-based Traffic Shaping for Stabilizing Oscillating Adaptive Streaming Players," *Proc. ACM NOSSDAV*, Oslo, Norway, Feb 2013, pp. 19-24.
[41] G. Zhang, J.Y.B. Lee, "On Data Wastage in Mobile Video Streaming," *Proc. IEEE WCNC*, Barcelona, Spain, Jun 2018, pp. 1-6.

Guanghui Zhang received his B.Eng. in Electronic Information Engineering from Shandong Normal University, Jinan, China, in 2013 and M.Sc. degree in Integrated Circuit and System from Peking University, Beijing, China, in 2016. He is currently a Ph.D. candidate in the Department of Information Engineering, the Chinese University of Hong Kong, where he participated in the research and development of multimedia technology.

Jack Y. B. Lee (M'95–SM'03) received his B.Eng. and Ph.D. degrees in Information Engineering from the Chinese University of Hong Kong, Shatin, Hong Kong, in 1993 and 1997, respectively. He is currently an Associate Professor with the Department of Information Engineering of the Chinese University of Hong Kong. His research group focuses on research in multimedia communications systems, mobile communications, protocols, and applications. He specializes in tackling research challenges arising from real-world systems. He works closely with the industry to uncover new research challenges and opportunities for new services and applications. Several of the systems research from his lab have been adopted and deployed by the industry.