Optimizing Hypervideo Navigation using a Markov Decision Process Approach

Romulus Grigoraş
IRIT UMR CNRS 5505
ENSEEIHT-Informatique
2 rue Camichel
Toulouse, France
grig@enseeiht.fr

Vincent Charvillat
IRIT UMR CNRS 5505
ENSEEIHT-Informatique
2 rue Camichel
Toulouse, France
charvi@enseeiht.fr

Matthijs Douze
IRIT UMR CNRS 5505
ENSEEIHT-Informatique
2 rue Camichel
Toulouse, France
douze@enseeiht.fr

ABSTRACT

Interaction with hypermedia documents is a required feature for new sophisticated yet flexible multimedia applications. This paper presents an innovative adaptive technique to stream hypervideo that takes into account user behaviour. The objective is to optimize hypervideo prefetching in order to reduce the latency caused by the network. This technique is based on a model provided by a Markov Decision Process approach. The problem is solved using two methods: classical *stochastic dynamic programming* algorithms and *reinforcement learning*. Experimental results under stochastic network conditions are very promising.

Categories and Subject Descriptors

G.3 [**Numerical Analysis**]: Probability and statistics—*Markov processes*; H.5.1 [**Information Interfaces and presentation**]: Multimedia Information Systems

General Terms

Markov Decision Process, Reinforcement Learning

Keywords

uncertainty, optimization, simulation, streaming, hypermedia, navigation, interaction, prefetching.

1. INTRODUCTION

The recent evolution of multimedia content (more complexity, more interactivity) and the widespread development of multimedia services (with interactivity, real-time constraints and multipoint delivery requirements) make it important to give the user a simpler and more efficient hypermedia experience by improving applications' adaptability. Beyond diverse technologies aiming at measuring, negotiating and guaranteeing various levels of QoS, the actual trend is to confront the problem of user satisfaction. Our approach is coherent with the work of the MPEG-21 [1] group.

In order to provide always more interactivity, ease of use and comfort, efforts need to be focused on:

- object coding, compression, content scalability,
- various levels of synchronisation (inter- or intra-media, group logical, temporal, discrete and continuous synchronisation etc.),
- specification of protocols for multimedia content and applications

The work presented here proposes a mechanism for dealing with "intelligent" multimedia content capable of learning *how we make use of it* or *how we have just made use of it* or *how we generally make use of it*. This learning process, based on the memorisation of user-content interactions, helps solving the problems mentioned above. They also help improving the QoS management (synchronisation, anticipation, hierarchical structuring) and provide interesting ideas for the description, indexing, searching and retrieval of multimedia content. Our approach benefits from previous works dealing with the prediction of user behaviour for adaptive web sites and learning video browsing behaviour (see for example [2], [12]).

The first two sections present some guidelines and related research. A first intuitive model of our problem is proposed in section 4 and significantly improved by a new model, described in section 5. The associated optimization problem can be solved as detailed in section 6. The paper finally presents some experiments and future work.

2. GUIDELINES

Our approach is based on the assumption that two similar users, going through a similar sequence of interactions with hypermedia documents, might have similar browsing intents. The system can use this to anticipate user actions and provide a more responsive navigating experience.

Figure 1: An elaborate hypermedia document for teaching computer architecture as experienced in our hypervideo player. The processor, enclosed in a clickable zone, is linked to a video segment that shows more details.

2.1 What interactions

Recording user interaction can be based on the observation of some chosen variables:

- Audio/video manipulation. Actions include stop/pause, slow motion access to certain images or sequences of images, user requests for enhanced/increased image quality (zoom, increased contrast), shared annotations of areas of interest, change of viewpoint etc. [11]
- Hypermedia navigation. Actions include forward/backward navigation, transfer interruption (cancel of the intent of following a link) etc.
- 3D interaction. Actions correspond to VRML sensors like touch, visibility, orientation etc.
- navigation inside panoramic environments. Actions can be zooming or adjusting orientation (QuickTimeVR [3]).

2.2 How to store the interactions

At least two different kinds of memory levels can be imagined:

- *Long-term memory* based on a (cumulative) statistical analysis of interactions of users from various communities or sessions. Long-term memory can also contain *influential events*. An influential event can for example be associated with a state that has never been crossed or from where no user has ever returned to the initial content. A link is called "influential" if, by following it, there is no way back to the initial content. The system can therefore refrain from preloading it and concentrate on fully loading the present content.
- *Short-term memory* refers to the most recent interactions of one or multiple users. For example, if one has just followed $n-1$ links on a page, it is likely that she will also follow the n^{th} link. Another example: it is very likely for a user to follow the first $n-1$ links on a given hypermedia document if other similar users (from the same virtual community or acting inside the same session) have just followed these links.

There are two ways to let the long- and short-term memory cooperate without redundancy:

- The short-term memory tracks apparently deviating behaviour compared to the most probable behaviour from the long-term memory.
- The short-term memory detects specific patterns in the actions performed by the current user or in the current session. The underlying intentions are to be discovered.

As it will be shown later, in both cases, the short-term memory is used to eliminate some limitations inherent to the long-term memory.

3. BEYOND THE WEB

3.1 Previous work

This study benefits from previous research in the field of predictive statistical modelling [6], which tackles two tasks: *user modelling* and *machine learning*. Most of the previous work has only dealt with hypertext systems [8, 9, 10]. The result of these research permits to distinguish two kinds of web agents (leading to active web sites [13]):

- recommendation systems that presents information or facilities to the user (they typically run off-line),
- systems that perform actions on behalf of the user (on- or off-line).

Several models have been suggested, the most widely used being discrete or continous Markov chains [8, 9, 10] and sequence mining [2, 8]. The latter seem to produce the best prediction for webpage requests.

Another interesting model using hidden Markov chains has been proposed in [11]. In this model, the browsing behaviour is characterized by a set of browsing states. The learning of browsing behaviour can be transformed into the problem of predicting the browsing by observing interaction sequences. In the HMM approach, the browsing states of a viewer are called "hidden states" while the sequence of user interactions corresponds to observation sequences.

The solution of a problem for the hypertext domain is not necessarily appropriate for hypermedia, especially if it is streaming. As opposed to web browsing, the presentation of streaming hypermedia must handle ressource consuming objects under tight temporal constraints. Nowadays, many web acces logs are available, but there are much less streaming hypermedia statistics (this is changing, though, as standards like SMIL and RTSP become more accepted).

Nevertheless, predicting future actions is not enough if we want to improve navigation smoothness. Appropriate actions should be taken as well and, in the case of streaming contents, this of course includes prefetching.

3.2 Our approach

A user browses through a multimedia hyperspace, and we want to predict what content she will access in the near future in order to provide her with the most fluid navigation. To achieve this, we must look for:

- a model that describes the use of the multimedia content (the possible interactions),
- the algorithms that will predict the future interactions and prefetch the corresponding content.

Prefetching reduces latencies[1] but also prevents bandwidth underusage. Latency is an important QoS factor that arises from two sources:

- network latencies, caused by the overload of servers or networks, or propagation delays,
- multimedia treatment latencies, that include the presentation startup delays (header parsing, decoding of the beginning of the stream, etc.)

This two types of latency cannot be fought without introducing costs. Prefetching reduces network latencies, but doing so aggressively causes higher network loads and burst traffic [2]. Prelaunching a hypermedia presentation lowers latencies, but introduces a heavy load on the client machine.

This article presents an efficient mecanism to reduce latencies of the first type. Further research on handling treatment latencies will be done in the future.

3.3 Hypervideos

The hypervideo [4, 5] is a sub-class of hypermedia based on the link-node structure of hypertext and composed of digital video (figure 2). Due to its time-based nature, hypervideo requires different aesthetic and rhetoric considerations than traditional static hypermedia. Hypervideo has the potential to be non-linear but, as opposed to text, it is non-static. Opportunities (represented by hyperlinks) come and go as the video sequences play. Like static hypermedia more than one opportunity can be presented at once, but unlike hypertext these opportunities go away if not selected.

These opportunities are presented in the form of hyperlinks, which can be of two types: *temporal* and *spatio-temporal*. Temporal links are presented in the form of annotations flowing on top of the video during a certain period of time. Spatio-temporal links take advantage of the hierarchical structure of the video and associate opportunities to particular objects at a certain time. For example, figure 1 presents a hypervideo used for teaching computer architecture and a hyperlink visible on top of the processor.

[1]One can easily observe that waiting several seconds to be able to play a RTSP-streamed content is not uncommon. This latency includes the time to connect to the server, to negotiate the transport parameters, to prefetch some of the data, and to startup the presentation.

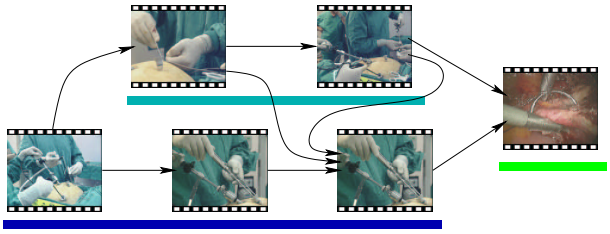


Figure 2: Example of a medical hypervideo containing two medium resolution sequences, and a short high resolution sequence.

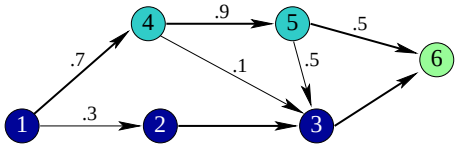


Figure 3: Coarse-state model.

3.4 What and when prefetching?

Let us analyse in which context prefetching is useful.

- We may want to prevent bandwidth under-usage, even if the associated cost is a limitating factor. This is the case, for example, when a hypervideo is made of streams whose bitrates are lower than the bandwidth. In this situation, it can be interesting to prefetch subsequent content when the currently played stream leaves some spare bandwidth. The observed latencies are low, but they can nevertheless be reduced.
- When some of the streams of the document (especially a hypervideo) have a bitrate greater than the available bandwidth. In this case, the only way to play the stream fluently is to bufferize an important amount of the data before launching the player. The latency introduced by such a process can be greatly reduced if the prefetch begins early enough.

4. INTUITIVE MODELS

We present a first rather naive model for hypermedia navigation. It is a simple way of taking into account past interactions [14].

4.1 A first Markov model

Intuitively, browsing through the hypervideo in figure 2 can be modeled with a 6-state graph. Taking into account the long-term memory (navigation statistics) enables us to weight each transition from a state to its successor with a given probability. This is shown in figure 3.

Nevertheless, other kinds of states, can be imagined ranging from finest to most rough. Coarse states (figure 3) show the logical structure of the content. In contrast, finer states (figure 4) better reflect the internal structure of the stream

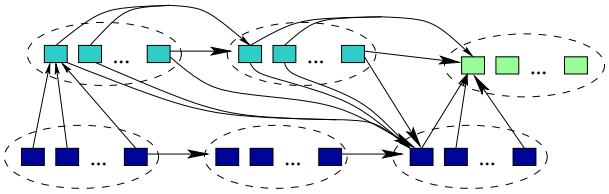


Figure 4: Fine-state graph for the hypervideo in figure 2.

state (i)	d_i (s)	br$_i$ (kb/s)	b_i (kb)
1	30	96	32
2	60		
3	70		
4	10	112	64
5	40		
6	5	192	96

Table 1: Bitrates (br), durations (d) and minimum sizes to be prefetched (b) of the hypervideo of figure 2.

(typically, one state = one GOP). For simplicity reasons, we chose a rough-states model for our experiments. Less states means a lighter and thus more mobile content, since the memory streams travel along with the content.

At the beginning of the navigation, only the long-term memory contains information. The short-term memory is empty. We use markovian prediction to find the sequence of states that are most likely to be visited.

We will take short-term memory into account by heuristically and dynamically modifying the transition probabilities. Therefore the states will no longer be Markov states in the strict sense (i.e. without memory).

Despite their simplicity, the use of Markov chains lets us easily show the interest of prefetching and heuristics.

4.2 A simple example

We use the 6-state graph of figure 3, with the characteristics presented in table 1.

Two prefetching policies are proposed:

- P* (*proportionnal*) policies. All states that can be reached are prefetched from the current state. The bandwidth allocated for each stream is proportional to the probability of the corresponding transition.
- B* (*best-first*) policies. Only the most probable state is prefetched.

Each policy has two versions:

- The *C (*conservative*) version tries to use as little bandwidth as possible, and stops prefetching as soon as the amount b (minimum size needed to begin playing) of the stream has been downladed.

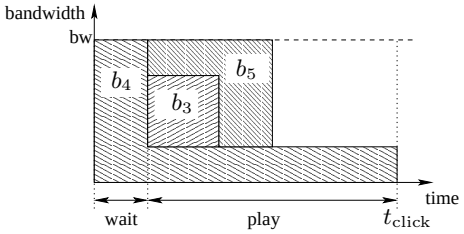


Figure 5: A PC policy applied to the current state 4, where the following states have probabilities .3 and .7.

Latency (in sec.)	No prediction	PC	PA	BC	BA
prefetch	3.75	2.55	0.99	2.25	0.85
	1.25+2.5	.8+1.75	.25+.74	.5+1.75	.25+.6
connexion	0.9	0.3	0.3	0.3	0.3
total	4.65	2.85	1.29	2.55	1.15

Table 2: Latencies depending on the static policy.

- The *A (*agressive*) version continues downloading and tries to use all available bandwidth.

Figure 5 illustrates a proportional conservative policy. At $t = 0$ is state 4, the buffers are empty, and thus, the user must wait for the amount b_4 to be downloaded. Next, while the video plays, the remaining bandwidth is used to prefetch, proporionnaly to their probabilities, the following states (5 and 3). Because the policy is conservative, prefetching stops as soon as the amount b_5 is available.

Table 2 shows the latencies computed for the path 1-4-5-6, supposing that the moment at which the user choses to follow a link is half-way the current video segment. For this path (which is the most likely to be followed), we observe a significant reduction of the latencies for the four policies. In order to understand why in these circumstances one policy gives better results than another, we present separately the total latency for states 1, 4 and 5 and that of state 6. During state 5, agressive policies manage to prefetch an important part of state 6's stream, whose bandwidth is greater than what is available.

The connexion latencies, presented separately (connexion row in Table 2), are not negligible[2]. It is worth observing that, even without prefetching, pre-initializing connexions alone is enough to reduce latencies significantly.

4.3 Shortcomings of the model

It is easy to observe that the example presented earlier does not help in evaluating the relevance of the elementary Markov model. The transition probabilities supplied by the long-term memory (in the general sense defined in 2.2) are not appropriate in the frequent case where the user comes back on an already-visited video segment.

Indeed, let us imagine that a segment s_1 has two successors

[2]Experiments realized with a QuickTime streaming server and a JMF/RTSP client show an average connexion time of 300 ms.

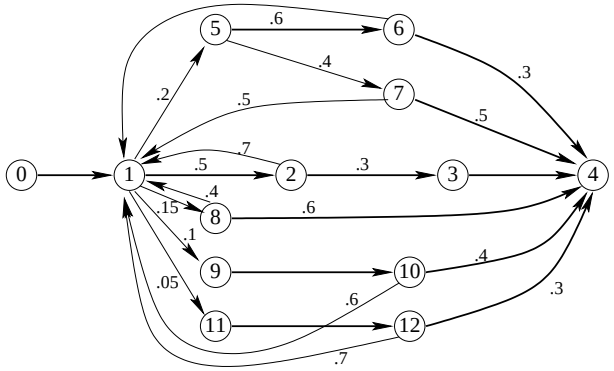
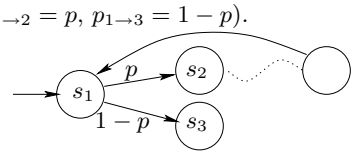


Figure 6: Example of a hypervideo containing a table of contents (state 1) pointing to several chapters.

f	heuristic	PC	PA	BC	BA
0.1		3.81	3.80	3.88	3.87
0.1	BETA	3.81	3.80	3.88	3.87
0.5		6.10	4.71	5.91	5.91
0.5	BETA	6.10	4.70	5.91	4.54

Table 3: Latencies computed for the graph in figure 6, with different prefetching policies and with/without the use of the BETA heuristic. The user clicks on a link at $f\times$ the length of the current segment.

s_2 and s_3 ($p_{1\to 2} = p$, $p_{1\to 3} = 1 - p$).



If a user follows the link to s_2, and comes back to s_1, the probability that she will re-follow s_2 is certainly lower than p. The long-term memory model is thus inadequate. In order to illustrate this and to demonstrate why it can be improved, we show the use of heuristics.

Heuristics consist in adaptively adjusting the transition probabilities and are based on the short-term memory. This modification is done by examining the path already followed.

In a more complex example (figure 6), we can illustrate how one of these heuristics works. For example, let's consider the path 0-1-2-1-5-7-1-8-1-9-10-1-11-12.

A heuristic (henceforth called "BETA") detects exhaustive navigation: after the user followed s_1's 3 first links (1→2, 1→5, 1→8), the links 1→9 and 1→11 are updated (multiplied by $\beta > 1$). This will stimulate the prefetching of states 9 and 11.

In conjunction with the PA policy, BETA introduces a small latency reduction. This reduction is more important if the policy BA is used (table 3).

This reduction also shows some of the weakness of the naive model. In fact, two random parameters are not taken into account:

- the moment the transition is made is not really considered (since the user is supposed to click at a given fraction of the video's duration),
- the average bandwidth is supposed to be constant.

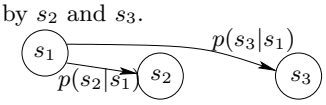
Another drawback of this model is that it is quite difficult to take into account cycles or back/forward actions in the recent navigation history, even if the use of heuristics can solve part of the problem.

5. A MORE POWERFUL MODEL

We introduce our new model by an example.

5.1 Illustrated introduction

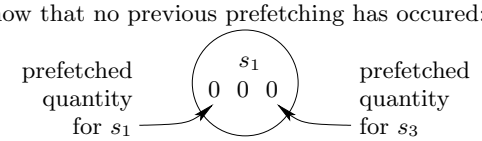
Let us consider a segment s_1 of duration d_1 and bitrate br_1, followed by s_2 and s_3.



The model previously introduced weights the link $s_1 \to s_2$ (resp. $s_1 \to s_3$) with the probability $p(s_2|s_1)$ (resp. $p(s_3|s_1)$), the probability to go to s_2 when we are in s_1 (resp to s_3 from s_1). Of course, $p(s_2|s_1) + p(s_3|s_1) = 1$.

Based on this, we build a new kind of state, called *buffer-state*, which encapsulates the current video segment, and the fill rate of each video segment's buffer. This fill rate depends on the previous loading decisions. We note a buffer-state as $\sigma = (s, r_1, ..., r_n)$, where n is the number of video segments and r_i the fill ratio of s_i's buffer.

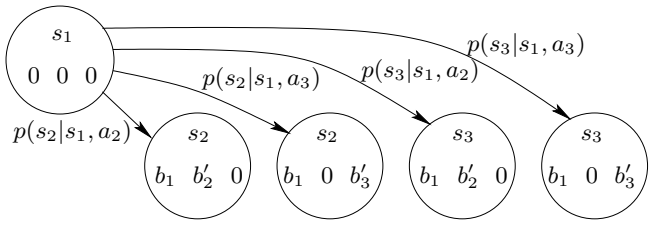
Thus, when we start playing the entry state s_1, the 3 empty buffers show that no previous prefetching has occured:



At the entrance of the buffer-state, the hypervideo player does the following:

- It downloads the minimum quantity of data needed to begin playing s_1 (a quantity called b_1). If bw is the average bandwidth, this causes a latency of b_1/bw.
- It decides whether, while playing the current state, it will also prefetch other video segments. In our model, we only consider "elementary" and disjoint prefetching actions: we can only prefetch (part of) b_i for one segment i, as opposed to the more sophisticated policies seen earlier (sequential, proportional, etc.). For our example, the possible actions are: a_2 = "prefetch s_2" and a_3 = "prefetch s_3".

Playing s_1 lasts at most d_1, and we consider here that the user clicks on the link chosing s_2 or s_3 at $d_1/2$. Thus, we obtain the following transition graph:

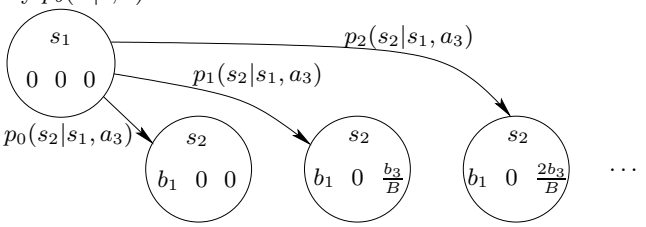


Where $p(s'|s, a)$ is the probability to go to s' when we are in s, and we chose action a at the start of s. In our case, we have $p(s'|s, a) = p(s'|s)$.

The b'_is are easily computed: $b'_i = \min(b_i, (\mathrm{bw} - \mathrm{br}_i) \times d_i/2)$. $\mathrm{bw} - \mathrm{br}_i$ represents the bandwidth available for prefetching while s_1 is read, that is during $d_1/2$.

At this point, our model encapsulates in its buffer-states the effect of all previous actions and transitions, that we could not have taken into account with the first model. Therefore, the buffer-state graph is closer to markovian conditions.

Now we can introduce the two random sources mentioned before (varying transition moments and varying bandwidth) into our model. For this, the fill-rate of each buffer is coded as a number between 0 and B. If we consider for instance the link $s_1 \rightarrow s_2$, and action a_3 (prefetch s_3 while playing s_1), we can go to $B + 1$ different buffer-states where a fraction $0, 1/B, 2/B, \ldots, b/B, \ldots 1$ of b_3 is prefetched. Each buffer-state transition is weighted by an unknown probability $p_b(s'|s, a)$.



This model indicates (approximatively, quantized on $B + 1$ levels) the quantity of data effectively buffered.

This model is not perfect, but:

1. it is simple,
2. back/forward navigation in the recent navigation history can be naturally considered by this model, as opposed to the previous one,
3. the two sources of randomness can be handled at the same level, which is theoretically and practically (for simulations) very appealing,
4. various other ideas could be included, especially the SMDPs [17], which handle non-discrete click times, but where the following state depends on the time spent in the current state,
5. the randomness concerning the bandwidth is the trickiest point anyway.

5.2 A Formal Markov Decision Problem

The prefetching of hypervideo segments can be seen as a *sequencial decision problem* under *uncertainty*, but *complete observability*. Indeed:

- decisions have to be made in sequence about whether a video segment is to be prefetched;
- the consequences of a decision are not known with certainty, because of unpredictable user behaviour and network conditions.

Thus, the formal MDP framework (*Markov Decision Process* [17]) can be used to model it. In a few words, a *MDP* is defined by a quintuple (S, A, T, P, R) where:

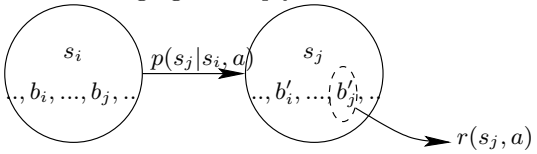
- S is the set of all possible *states* of the system;
- A is the set of all the *actions* that can be applied to it;
- T is the ordered set of *instants* at which decisions can be made, that is the *global temporal horizon*;
- P defines the *transition probabilities* between any pair of states in S after executing an action in A;
- r defines the *local rewards* associated with these transitions.

In this framework, the usual request is to compute what is called an *optimal policy*, that is a function π^* that associates with any state $\sigma \in S$ and any *time* $t \in T$ an optimal action $\pi^*(\sigma, t)$, that is an action that maximizes the expected *global reward* on the remaining temporal horizon. This global cost or reward may be defined as the *sum*, the *discounted sum*, or the *mean value* of the local costs or rewards associated with the actual transitions. When the optimal policy does not depend on the time t and only depends on the state s, it is said to be *stationary*.

For our problem:

- S is the set of buffer-states,
- A contains the possible prefetch actions,
- T is $\mathbb{N}$. The instants are given by the order of the click in the navigation path,
- P is defined by a set of *random variables* associated with the possible transitions from a buffer-state to another,
- $r(\sigma, a)$ is defined as the *decrease in latency* as compared to a the dummy no-preloading policy. For instance, in the example of 5.1 $r((s_1, r_1, r_2, r_3), a_2) = (1 - r_2)b_2/\mathrm{bw}$. In fact, the reward does not depend on the action.
- the *global reward* is the sum of all gains that the user benefits from while she is browsing a hypervideo.

The following figure simply resumes our model.



At the beginning of s_i we decide a. Both user's click and bandwidth randomness influence the filling levels of buffers corresponding to the states s_i and s_j. These buffer changes are contained by s_j. We associate a reward with the state s_j, reward that is naturally proportional with the buffer level b'_j.

6. OPTIMIZATION FOR MDPS

6.1 Stochastic dynamic programming

The MDP theory associates with each policy π a *value function* V_π that maps a state $\sigma \in S$ and a time $t \in T$ to the expected global reward $V_\pi(\sigma, t)$, obtained by applying π from σ and t. In our case, we use:

$$V_\pi(\sigma) = r(\sigma, \pi(\sigma)) + \gamma \sum_{\sigma' \in S} p(\sigma'|\sigma, a) V_\pi(\sigma')$$

where $\gamma \leq 1$.

We can interpret this value function as the sum of two terms: the reward associated with the current state (and action) and the γ-weighted mean of future rewards, while following π.

Bellman's optimality equations (equations 1 [15]) characterize in a compact way the *unique optimal value function* V^* from which an *optimal policy* π^* can be straightforwardly derived. In case of a stationary policy and a global reward defined as the discounted sum of the local rewards, these equations become:

$$V^*(\sigma) = \max_{a \in A} (r(\sigma, a) + \gamma \sum_{\sigma' \in S} p(\sigma'|\sigma, a) V^*(\sigma')) \quad (1)$$

and

$$\forall \sigma \in S \quad \pi^*(\sigma) = \operatorname*{argmax}_{a \in A} (r(\sigma, a) + \gamma \sum_{\sigma' \in S} p(\sigma'|\sigma, a) V^*(\sigma'))$$

6.2 Value iteration

For MDPs with tiny finite S and A sets, *value iteration* or *policy iteration* are dynamic programming algorithms. They exploit efficiently the problem data and structure, in order to compute this optimal value function and to derive an associated optimal policy. In our case, S and A are finite, but S is quite large.

We want to solve Bellman's equation in $V^*(\sigma)$ with an iterative method that computes a series $(V_n)_n$. We chose an arbitrary V_0, and we iterate:

$$\forall \sigma \quad V_{n+1}(\sigma) = \max_a \left(r(\sigma, a) + \gamma \sum_{\sigma'} p(\sigma'|\sigma, a) V_n(\sigma') \right)$$

If the rewards are bounded, the series converges to V^*, and we can deduce π^* [17].

In brief, each iteration improves the current policy associated with V_n.

6.3 Reinforcement Learning

The Artificial Intelligence community has recently developed *reinforcement learning* methods that make the optimization of policies for large MDPs possible, by means of simulations and approximations of the value function and/or of the policy.

The reinforcement learning approach consists in learning an optimal policy by estimating iteratively the optimal value function of the problem on the basis of simulations. Today, reinforcement learning is one of the main approaches able to solve sequential decision problems with unknown transition probabilities and/or MDPs with large state spaces. Various reinforcement learning algorithms exist, such as *Q-learing*, *R-learing*, *Sarsa*, and others [18].

In this paper, we consider the *Q-learning* algorithm [16], that can be used when the transition probabilities of the MDP are not known, as it is the case in our problem.

Without keeping a process model, Q-learning is a reinforcement algorithm allowing the resolution of Bellman's equation for the γ-weighted criterion. It is commonly used in practice because of its simplicity. Its principle consists in updating iteratively the values of the V^* function we search, observing the instaneous transitions and their corresponding revenue.

The fixed-point form of Bellman's equation is unusable to design an adaptive resolution algorithm. Thus, Watkins [17] introduced a value function Q that carries the same information as V.

For a given policy π and the value-function V_π, we define the new function

$$\forall \sigma \in S, a \in A \quad Q_\pi(\sigma, a) = r(\sigma, a) + \gamma \sum_{\sigma'} p(\sigma'|\sigma, a) V_\pi(\sigma')$$

Q_π's value can be understood as the expected value of the criterion for the process that starts in σ, executes action a and thereafter follows policy π. It is clear that $V_\pi(x) = Q_\pi(x, \pi(x))$, and the Bellman equation that Q^* verifies becomes:

$$\forall \sigma \in S, a \in A \quad Q^*(\sigma, a) = r(\sigma, a) + \gamma \sum_{\sigma'} p(\sigma'|\sigma, a) \max_b Q^*(\sigma', b)$$

We then have

$$\forall \sigma \in S \quad V^*(\sigma) = \max_a Q^*(\sigma, a) \quad \pi^*(\sigma) = \operatorname*{argmax}_a Q^*(\sigma, a)$$

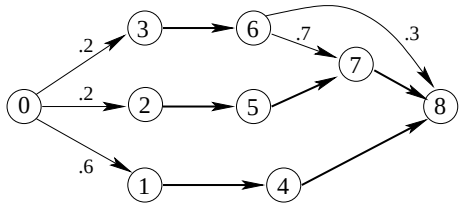
The principle of the Q-learning algorithm (figure 7) is to update the current value-function Q_n for the values (σ_n, a_n) at transition $(\sigma_n, a_n, \sigma_{n+1}, r_n)$ (where σ_n and σ_{n+1} are the successive states, a_n is the action taken and r_n the local reward).

In this algorithm, we fix the total number of iterations N_{tot} initially. The *learning rate* $\alpha_n(\sigma, a)$ decreases towards 0 at each iteration. The `simulate` function returns a new state and the associated reward according to the system's dynamics. The initialization function `initialize` just sets all Q_0's values to 0.

The choice of next state to be explored (`choseState`) is only made among the states reacheable from the current state. In the same way, only states that are within reach can be chosen (`choseAction`) to be prefetched.

Initialize Q_0
for $n = 0$ **to** $N_{\text{tot}} - 1$ **do**
  σ_n =`choseState`
  a_n =`choseAction`
  (σ'_n, r_n) =`simulate`(σ_n, a_n)
  /* update Q_{n+1} */
  $Q_{n+1} \leftarrow Q_n$
  $d_n = r_n + \gamma \max_b Q_n(\sigma'_n, b) - Q_n(\sigma_n, a_n)$
  $Q_{n+1}(\sigma_n, a_n) \leftarrow Q_n(\sigma_n, a_n) + \alpha_n d_n$
end for
return $Q_{N_{\text{tot}}}$

Figure 7: The Q-learing algorithm.



state (i)	d_i (s)	br_i (kb/s)	b_i (kb)
0	30	96	32
1	60		
2	70	64	
3	10	112	64
4	40		
5	60		
6	60		
7	70		
8	20	128	128

Figure 8: A more complex graph, and the associated bitrates (br**), durations (**d**) and minimum buffer to be prefetched (**b**).**

If α_n decreases fast enough towards 0 and if we visit long enough the $S \times A$ space (that is N_{tot} is big enough), the Q-learning algorithm converges *almost surely* (with a probability of 1). By using Q-learning, we can consider much larger models, at the expense of an increased number of iterations.

7. EXPERIMENTS

We have used the hypervideo described by figures 8 and 9, in conjunction with the Value Iteration and the Q-learning algorithms.

7.1 Experimenting with Value Iteration

The implementation of the Value Iteration algorithm requires 3 steps:

Model generation The value iteration algorithm takes advantage of the transtion probabilities between buffer states. In order to learn these probabilities, we have simulated 100000 random navigations. In our example there are $n = 9$ states, and the buffer granularity is $B = 4$. Thus, there are

$$N = nB^n$$

buffer states, that is approximately 2×10^6 states. We keep the transition probabilities in a three-dimensional matrix of size $N \times N \times n = 15 \times 10^{13}$. Each element $p(\sigma, \sigma', a)$ represents the probability to go to σ' from σ having chosen a for the duration of σ. Fortunately, the matrix is sparse, as the number of visited states is small (about 3300). Cycles in the macro-state graph do not imply an explosion of the number of possibly visited buffer-states. In fact, the buffer levels accumulate until the full level is reached.

Resolution Using the model described above and the value iteration algorithm, we identified the optimal policy for the considered hypervideo, that is, the optimal prefetching action to be taken for each buffer state.

Validation We validate this optimal policy by comparing it to other policies.

Buffer levels are influenced by the available bandwidth and by the length of the buffer-state. These parameters are simulated as follows:

- The average bandwidth during the current state is uniformly distributed between a minimum and a maximum value. In this example, we chose $\text{bw}_{\min} = 96$ kb/s and $\text{bw}_{\max} = 128$ kb/s but any other network model could be easily integrated.
- The duration of a buffer-state is uniformly distributed between 0 and d_i. We measure this duration from the moment the video segment begins playing. This is because, while in the buffering stage, the user is not presented the link and thus a state change cannot occur.

Table 4 (corresponding to figure 8) presents the results of 1000 simulated random paths, with an average length of 4.22 hops. Both the average latencies and standard deviation are shown in three different cases: the original graph (0), when s_8 requires a large buffer (1), and when it demands an even larger buffer (2).

The figures show an important reduction of the latencies as well as the standard deviations if the optimal policy π^*_{vi} is used, compared to the best-first policy π_{BC}. The difference with the case with no prefetching at all (dummy π_0) is even more significant.

In fact, if we consider the sample paths chosen for case 1, π^* tends to prefetch the heavy s_8 very early. This illustrates the shortsightedness of π_{BC}.

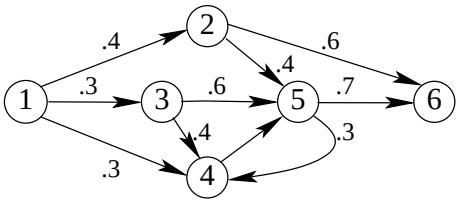
Another strong quality of the optimal policy is that, even if it did not anticipate the path the user actually chose, it is able to provide the next optimum action at every step.

7.2 Experimenting with Q-learning

Our experiments with Q-learning consisted of only 2 steps: the resolution and the validation, as Q-learning does not require to maintain a model. As shown in the table 4, the results for 1000 simulated random path using the optimum

no.	modifications	example paths		latencies			
		path	optimal actions	π^*_{vi}	π_{BC}	π_0	π^*_{ql}
0	$b_8 = 128$	0-1-4-8	1,4,8	1.55	1.90	2.90	1.70
		0-2-5-7-8	1,5,7,8	0.56	0.73	0.34	0.40
1	$b_8 = 360$	0-1-4-8	8,4,8	2.28	3.96	5.04	2.30
		0-3-6-7-8	8,6,7,8	1.11	1.59	0.42	1.11
2	$b_8 = 720$	0-1-4-8	8,4,8	4.15	7.36	8.31	4.30
		0-3-6-7-8	8,6,7,8	1.15	2.60	0.62	1.16

Table 4: This table sums up the results of 3 experiments, where the "weight" of s_8 was modified. For each situation, we apply 4 prefetching policies to 1000 random browsing paths, and indicate the average and standard deviation (below) of the latencies.



state (i)	d_i (s)	br_i (kb/s)	b_i (kb)
1	40	64	64
2	10		
3	10	32	32
4	20	96	196
5	5	64	32
6	20	96	196

	π_0	π_{BC}	π^*_{ql}
$\bar{x}$	4.57	3.14	2.13
σ	0.81	0.87	0.62

Figure 9: A graph with more links and the observed latencies.

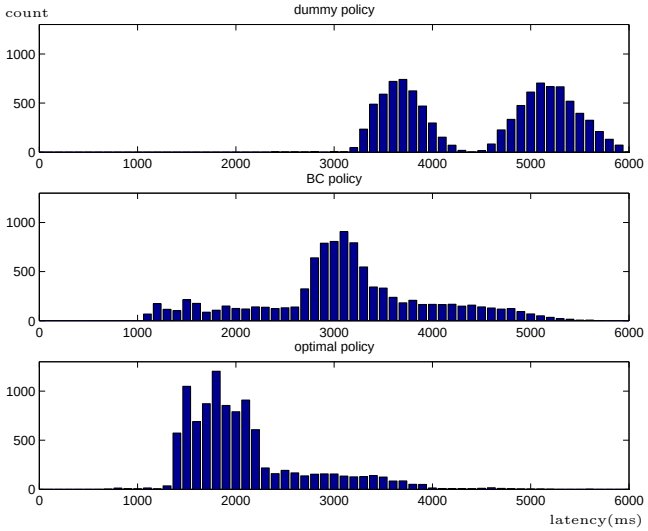


Figure 10: Latencies' distribution for three policies (π_0, π_{BC}, π^*) for a graph in which the optimal policy performs clearly much better than the best-first strategy.

policy π^*_{ql} are very similar to those obtained by Value Iteration. We chose $N_{\text{tot}} = 100000$.

In order to better understand the quality of the optimum policy we consider the graph from figure 9. Each histogram of figure 10 shows a distribution of observed latencies for 10000 simulated random paths. One can easily notice the incremental improvement from dummy policy through best-first policy to the optimal one, as computed by the Q-learning algorithm. The first histogram (latencies obtained without any policy) clearly identifies the two heavy states that have an important effect on latencies.

As expected, this shows the most important benefit of the optimum policy: the reduction of latencies' variability.

7.3 Implementation

We have started the implementation of HyVideo (figure 11), a client/server for hypervideo navigation. The client is based on a JMF/RTSP player. It uses a prediction unit that downloads the optimum policy from the prediction sever, instructs the buffer management unit which buffers to fill, and the player pool which players to prestart. At the end of each navigation, history information (path, observed bandwidth and latencies) is sent back to the prediction server. The streaming is done by a RTP/RTSP server that adjusts the bitrate of the streams according to network parameters as computed from RTCP feedback.

7.4 Solving very large problems

Despite the sparcity of the matrix used by either the Value Iteration or the Q-learnig algorithms, solving very large problems(tens of video segments, fine buffer granularity or complex policies) requires considerable ressources, in both memory and processor power. Moreover, given the fact that the client is supposed to download the optimum policy vector, we cannot afford it to be very large.

An alternative approach for solving large MDPs consists in using an *a priori* parametrized policy, defined by experts, and optimizing the values of these parameters through the use of simulated trajectories. Note that the Markovian feature of the problem is no longer exploited in this approach and that the whole process is considered as a "black box" that transforms input parameters (the policy parameters) into an output criterion (the expected global cost or reward).

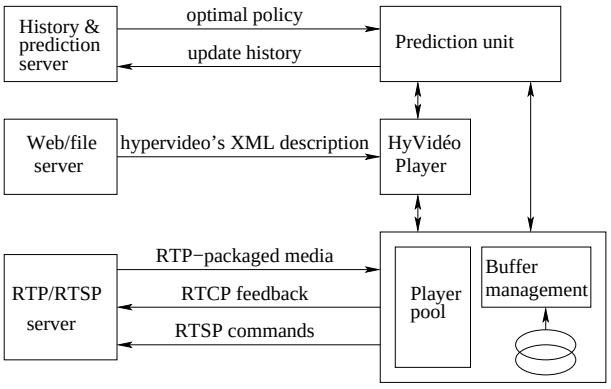


Figure 11: Architecture of the HyVideo Platform

8. CONCLUSION

We have shown that reducing latencies in the context of hypervideo navigation is possible using prefetching policies.

Optimal policies in the sense of our markovian model can be obtained automatically. The strength of our obtained policies is that they reduce both the latency and its variation while dealing with stochastic network conditions.

Both theoretical and practical perspectives can be conceived. It would be interesting to consider the other strong point of reinforcement learning, that is value functions approximation. This would eventually help to drastically reduce the memory requirements of the algorithm and allow us to solve very large problems.

In the same spirit, the full implementation of our software architecture dealing with self-adapting content raises a number of interesting issues, including:

- the quality of the prediction under special network conditions (steep increase/decrease of the available bandwidth) and TCP/IP friendliness,
- the influence of the buffer granularity over the quality of the prediction,
- the possible use of more complex policies, similar to the ones presented in the first intuitive model,
- the scalability of the system, in terms of both the number of simultaneous connexions and the total bandwidth used by these connexions.

Finally, we are interested in further extending our model by taking into account some other aspects of user modeling, such as sequence mining and intension browsing. We have started experiments with a hypervideo used for teaching computer architecture to a student population (figure 1).

9. REFERENCES

[1] MPEG-21, MPEG Group, *Multimedia Framework*, `http://www.cselt.it/mpeg`

[2] B. Trousse, *Evaluation of the Prediction Capability of a User behaviour Mining Approach for Adapative Web Sites*, Proc. RIAO 2000, 6th Conference on Content-Based Multimedia Information Access, Paris, France, 2000.

[3] Apple, QuickTimeVR, `http://quicktime.apple.com`

[4] N. Sawhney, D. Balcom, I.Smith, *Authoring and navigating video in space and time: A Framework and Approach towards Hypervideo*, IEEE Multimedia, 4, 4, 1997 `http://www.lcc.gatech.edu/gallery/hypercafe`

[5] J. Tolva, *MediaLoom, An Interactive Authoring Tool for HyperVideo*, M.S. Project Paper, Georgia Institute of Technology, 1998.

[6] I. Zukerman, D. Albrecht, *Predictive Statistical Models for User Modeling* User Modeling and User-Adapted Interaction, 2001.

[7] M. Crovella, P. Barford, *The Network Effects of Prefetching*, Proc. of Infocom'98, IEEE, April 1998.

[8] Y. Aumann et al., *Predicting Event Sequences: Data Mining for Prefetching Web-pages*, Proc. of KDD'98.

[9] K. Aberer, S. Hollfelder, *Resource Prediction and Admission Control for Interactive Video Browsing Scenarios Using Application Semantics*, GMD Report No. 40, September 1998.

[10] D. Albrecht, I. Zukerman, A. Nicholson, *Pre-sending Documents on the WWW: A comparative Study*, Proc. IJCAI'99, August 1999.

[11] T. Syeda-Mahmood, *Learning and Tracking Browsing Behavior of Users using Hidden Markov Models*, IBM Make It Easy Conference, 2001.

[12] T. Syeda-Mahmood, D. Ponceleon, *Learning video browsing behavior and its application in the generation of video previews*, Proc. ACM Multimedia 2001.

[13] A. Kohrs, B. Merialdo, *Creating user-adapted Websites by the use of collaborative filtering*, Interacting with Computers, Volume 13, Issue 6, August 2001.

[14] R. Grigoraş, V. Charvillat, M. Douze, *Self-adaptive multimedia content*, Proc. ECMCS, September 2001.

[15] R. E. Bellman, *Dynamic programming*, Princeton University Press, 1957.

[16] D. P. Bertsekas, J. N. Tsitsikis, *Neurodynamic programming*, Athena Scientific, Belmond, Massachusetts, 1996.

[17] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*, Wiley-Interscience, New York, 1994.

[18] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, Massachusetts, 1998.