

# Automatic Resource-aware Construction of Media Indexing Applications for Distributed Processing Environments

Ole-Christoffer Granmo

Department of Informatics, University of Oslo  
P.O. Box 1080 Blindern, N-0314 Oslo, Norway  
olegr@ifi.uio.no

<http://www.ifi.uio.no/~olegr>

**Abstract.** Constructing pattern classification based media indexing applications for distributed processing environments (DPEs) often requires knowledge in feature extraction as well as distributed processing. In order to achieve an appropriate tradeoff between indexing error rate and processing time, feature extractors (FEs) must be carefully selected based on their application specific indexing and DPE specific processing characteristics. In this paper we propose an algorithm which given a trainable pattern classifier, a generic FE library, a DPE (within a certain class of DPEs) and a manually indexed media stream, automatically generates a sequence of media indexing application configurations. The algorithm is based on task graph scheduling and feature subset selection, combined in a heuristic search procedure. The media indexing application configurations are generated in increasing order of indexing error rate and decreasing order of processing time. As a result, media indexing applications with various tradeoffs between indexing error rate and processing time can be constructed automatically.

## 1 Introduction

The technical ability to generate volumes of digital media data is becoming increasingly “main stream” in today’s electronic world. To utilize the growing number of media sources, both the ease of use and the computational flexibility of methods for content-based access must be addressed; e.g. an end-user may want to access live content automatically under a variety of processing environments ranging from complex distributed systems to single laptops.

In order to make media content more accessible, pattern classification systems which automatically classify media content in terms of high-level concepts have been taken into use. Roughly stated, the goal of such pattern classification systems is to bridge the gap between the low-level features produced through signal processing (feature extraction) and the high-level concepts desired by the end-user. Automatic visual surveillance [1], indexing of TV Broadcast News [2] and remote sensing image interpretation [3] are examples of popular application domains. For instance, in [2], a TV Broadcast News video stream is automatically parted into segments such as Newscaster, Report, Weather

Forecast and Commercial. Thereby, the video stream is indexed and an end-user can access the different types of segments directly, without skimming through the complete video stream.

Recent media indexing applications are based on pattern classifiers which can be trained automatically from manually indexed media streams given a set of feature extractors (FEs). In short, such approaches do not require manual specification of the correspondence between the low-level features produced by FEs and the high-level concepts desired by the end-user. Consequently, constructing media indexing applications becomes more flexible and accessible.

Still, the FEs must be carefully selected based on the desired level of misclassification (error rate) as well as the desired processing time of the media indexing application. FEs may differ in

- computational complexity,
- dependencies on other FEs (restricts parallelization),
- the size of the features produced,
- their relevance to the high-level concepts of interest,

and as a result, manual selection of FEs often requires expert knowledge in feature extraction and distributed processing.

To lessen the difficulty of selecting FEs which are relevant to the high-level concepts of interest, FE subset selection approaches from machine learning and pattern recognition can be applied [4]. However, such approaches do not consider the processing characteristics of FEs nor the tradeoff between error rate and processing time in a distributed processing environment.

In this paper we introduce the ARCAMIDE algorithm, a general purpose algorithm for Automatic Resource-aware Construction of Applications for Media Indexing in Distributed processing Environments. In short, the algorithm automates the selection of FEs given

- the number and location of available CPUs,
- the communication latency and bandwidth,
- a trainable classifier,
- a set of manually indexed media streams.

The overall goal of the ARCAMIDE algorithm is to produce a sequence of FE configurations in increasing order of error rate and decreasing order of processing time. An FE configuration, in combination with the corresponding trained classifier, form the basis of the constructed media indexing application. The constructed media indexing application can be run on a distributed feature extraction and classification architecture, such as the distributed component-based DMJ-framework [5] or the multi-agent based framework in [6].

The rest of the paper is organized as follows. In Sect. 2 we briefly describe the concept of training based media indexing and error rate estimation. Additionally, we describe our media indexing application case. Then, in Sect. 3 we introduce an FE library which in addition to FEs contains FE processing characteristics. In Sect. 4 we define the class of distributed processing environments we are targeting. Based on the

FE processing characteristics and the defined class of distributed processing environments, we describe an algorithm for scheduling a media indexing application on CPUs in Sect. 5. The overall ARCAMIDE algorithm which integrates the error rate estimation technique and scheduling algorithm is proposed in Sect. 6. Empirical results are presented in Sect. 7. Finally, we conclude in Sect. 8 and discuss further work.

## 2 Media Indexing

A typical media indexing application consists of a set of *feature extractors* (FEs) and a *classifier*. When indexing a media stream, the FEs produce a set of features for each media segment (part) in the stream. The features quantify characteristics of the media segments such as the roughness of the texture in an image region or the amount of motion between two images. The goal of the classifier, then, is to associate a high-level concept to each media segment based on the produced set of features. Finally, the media segments are indexed with the determined concepts.

In order to be able to automatically associate high-level concepts to media segments from the produced features, a classifier can be trained on manually indexed media streams. Typically, the training is based on finding a more or less accurate mapping between feature space and concept space, within a hypothesis space of possible mappings defined by the classifier type (e.g. 5-state Hidden Markov Model, 10-state Hidden Markov Model, Decision Tree, Neural Network, etc.).

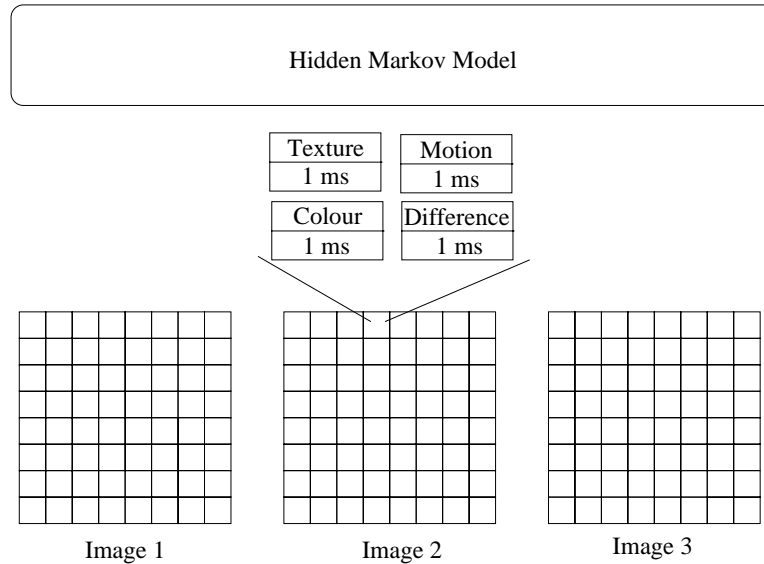
After training the classifier on manually indexed media streams (training set), the classifier can be evaluated by measuring the number of misclassifications on another manually indexed media stream not used for training (test set). We shall by the *estimated error rate* of a classifier mean the number of misclassifications divided by the total number of classifications on the test set. This estimated error rate can be seen as a measure on how accurately the classifier will index novel media streams.

Depending on the media indexing application, various levels of error rate may be allowable. For instance, misclassifying events when monitoring an airport for security reasons may be more critical than misclassifying events when indexing a Baseball video stream. Furthermore, when a classification error occurs, the consequences of different types of errors may have different costs. E.g. it may be more costly to misclassify a friendly airplane as hostile, than to classify an enemy airplane as friendly (depending on the situation).

The frequency of each type of error may be traded off against each other based on their cost. Furthermore, when selecting FEs the error rate may be traded off against processing time. In the next sections we focus on the latter. The integration of the former will be the focus of our further work. For a thorough treatment on training and evaluating classifiers, readers are referred to the literature (for example [7] and [8]).

We now introduce our media indexing application case, which is used to exemplify various aspects of the ARCAMIDE algorithm. The media indexing application case consists of a video camera capturing the traffic on a road from ground level. Each image in the image stream from the camera is parted into 8x8 image regions. A texture feature, a colour feature and a difference picture based motion feature can be extracted in each image region. Extracted features are fed to a collection of Hidden Markov Models

(HMMs) for classification. This is illustrated in Fig. 1. The goal of the media indexing



**Fig. 1.** A stream of images from a camera where each image is parted into 8x8 image regions. A texture feature, a colour feature and a difference picture based motion feature are extracted and fed to a collection of Hidden Markov Models

application is to detect traffic events such as red bus passing to the left, red bus passing to the right, collision, bicycle passing to the left/right, etc.

In this paper we focus on detecting vehicles passing in each direction. We construct a training set by manually identifying a set of image sequences where a vehicle passes from the left to the right and a set of image sequences where a vehicle passes from the right to the left. It is assumed that only one vehicle appears at a time in the camera view in order to keep the HMM state space and the number of HMMs at a practical level. We train one HMM for each image sequence set. During training the HMMs learn how a vehicle can be detected from the extracted features, and how a vehicle moves from image to image in the respective direction.

Obviously, the processing time and the error rate of our media indexing application depend on which FEs are used. For the task above the motion FEs may be highly relevant, and the texture FEs and the colour FEs may be of some relevance. Furthermore, features extracted from image regions not centered on the possible vehicle paths may be irrelevant. When using all the FEs, the minimum error rate and the maximum processing time can be achieved. The processing time can be reduced by using fewer FEs, possibly at the cost of increased error rate.

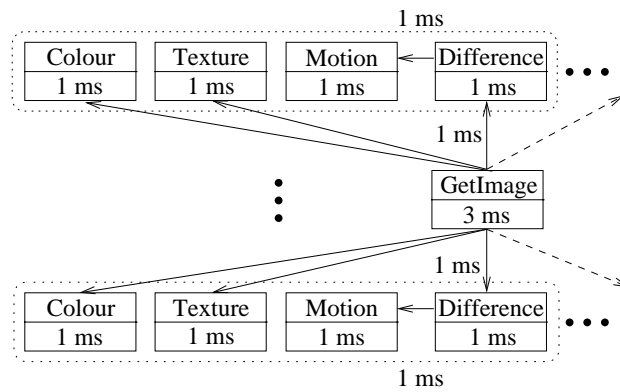
### 3 Feature Extractor Library

We assume that the FEs are organized in a library where their processing characteristics are specified. The following processing characteristics are used:

- a set of directed acyclic graphs (processing DAGs) where a node corresponds to an FE, and an edge between two nodes corresponds to the communication of a feature between the two respective FEs,
- the estimated processing time of each FE in milliseconds on various CPUs,
- the size in bytes of each feature communicated along an edge.

The processing time of each FE and the communication time of each feature can be estimated based on the above processing characteristics when the allocation of FEs to CPUs as well as the communication latency and bandwidth between FEs are given.

Fig. 2 is an excerpt from our example media indexing application processing DAG, annotated with processing times and communication times. The processing DAG spec-



**Fig. 2.** A processing DAG. Each group of FEs (indicated by the dotted boxes) is related to an image region. There are in total 64 FE groups

ifies that the image must be extracted from the image stream before the colour, texture and difference picture feature can be extracted in an image region. Furthermore, the difference picture in an image region must be extracted before the motion feature can be extracted. It takes 1 millisecond to communicate an image region from the image extractor to for instance a texture extractor. Furthermore, it takes 1 millisecond to extract the texture feature.

Let *entry FEs* be FEs without parents and let *exit FEs* be FEs without children in the processing DAGs. In the above context, it is assumed that the classifier only uses the features produced by exit FEs. Hence, the classifier can be seen as the final processing DAG node, connecting all the FE processing DAGs. Due to this role, the classifier will be treated as a special case throughout the paper.

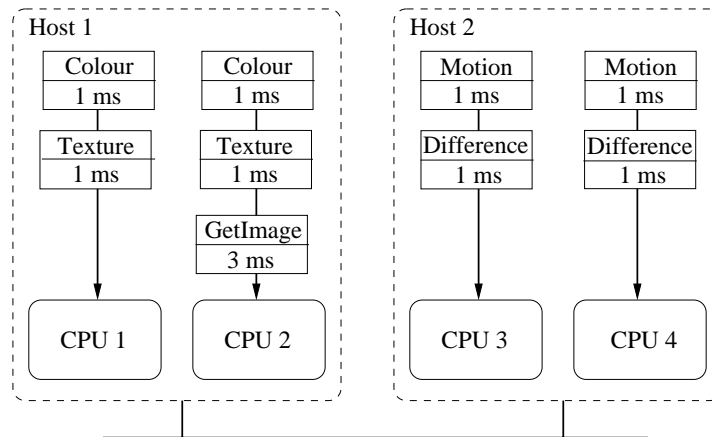
## 4 Distributed Processing Environment

Generally, a media indexing application is run on one or more interconnected CPUs in a distributed processing environment (DPE). When run on multiple CPUs the processing time of the media indexing application can often be sped up by extracting features in parallel. In this paper we consider DPEs consisting of homogeneous CPUs where communication between two CPUs is

- contention free,
- limited by a constant latency and bandwidth.

These simplifications are introduced to avoid the additional complexity caused by heterogeneous DPEs, communication contention, routing, etc. (which are not the focus of this paper) while still handling a significant class of DPEs (e.g. dedicated homogeneous computers connected in a dedicated switched LAN).

The classifier and each FE is allocated to a CPU where they are executed sequentially in a predefined order. When an FE is to be executed, the respective CPU sleeps until the features produced by the processing DAG parents of the FE are received. When the classifier is to be executed, the respective CPU sleeps until the features produced by exit FEs are received. For instance, the FEs from Fig. 2 can be allocated and ordered as



**Fig. 3.** A distributed processing environment. The FEs from two image regions are allocated to four CPUs and ordered for execution

shown in Fig. 3 when 4 CPUs are available. GetImage is executed before Texture which is executed before Colour, on CPU 1. CPU 1 and CPU 2 finishes processing after 5 milliseconds (here it is assumed that the communication time between CPUs on the same host is insignificant). CPU 3 and CPU 4 finishes processing after 6 milliseconds as 1 millisecond is spent communicating image regions from CPU 2 to CPU 3 and CPU 4.

Hence, the processing time of the given processing DAGs is 6 milliseconds (the amount of time elapsed from the execution of the processing DAGs starts until all the nodes have been executed).

## 5 Processing Time

The processing time of a set of FEs and a classifier (the processing DAG) determines the number of media segments which can be indexed per second. When doing *online* media indexing the number of media segments which can be indexed per second determines the sampling frequency. According to the Nyquist sampling theorem [9], any function of time  $f(t)$  (e.g. stream of high-level concepts) whose highest frequency is  $W$  can be completely determined by sampling at twice the frequency,  $2W$ . In other words, if a stream of high-level concepts is sampled at a frequency less than twice the frequency of the finest details in the stream, high-level concepts may be missed. When doing *offline* media indexing the number of media segments which can be indexed per second determines the amount of time necessary to index a given media stream.

In this section we describe a simple scheduling and processing time estimation algorithm targeting the class of processing DAGs defined in Sect. 3 and the class of DPEs defined in Sect. 4. The algorithm is based on basic task graph scheduling [10]. In short, it allocates and schedules a set of FEs and a classifier on CPUs (the goal is to minimize the estimated processing time given the DPE), before the estimated processing time is returned. The algorithm is summarized in Fig. 4 and described below.

We define the *b-level* of an FE to be the length of the longest path from the FE to an exit node. Likewise, the *s-level* of an FE is the length of the longest path from the FE to an entry node. The *length of a path* is simply the sum of the FE processing times (specified in the FE library) on the path. When calculating the b-level or the s-level of an FE the FE itself is included in the path.

An FE is either *allocated* to a CPU ( $A$ ) or *not allocated* to a CPU ( $N$ ). Initially, none of the FEs are allocated to CPUs. At each iteration of the processing time algorithm, the non-allocated FE with the largest b-level is allocated to a CPU. This means that execution of long processing DAG paths are prioritized before execution of short processing DAG paths. The main reason behind this strategy is that the longest processing DAG paths often determine the processing time of the processing DAGs when multiple CPUs are available, and accordingly should be executed as early as possible. As an example of the above strategy, in Fig. 2, the b-level of *GetImage* is 5 milliseconds, *Difference* 2 milliseconds, and *Colour* 1 millisecond. Consequently, if *GetImage* has been allocated to a CPU, *Difference* FEs are allocated to CPUs before *Colour* FEs.

When an FE is allocated to a CPU the FE is scheduled at the earliest *start time* possible. The FE may be started when

- the CPU becomes available after previous processing
- the FE receives the features produced by its processing DAG parents.

The scheduled *stop time* of an FE is simply the sum of its scheduled start time and its processing time (specified in the FE library). An FE receives a feature from a processing DAG parent at the scheduled stop time of the parent if the two FEs are located on the

---

```

; The processing time algorithm
pt(F)
; Initially all FEs are unallocated
A := ∅;
N := F;

WHILE #N > 0 DO
; Identifies FE with largest b-level
n_max := ARGMAX n IN N: b_level(n);
; Identifies the CPU which allows the earliest start time
p_min := ARGMIN p IN P: start_time(n_max, p);
; Allocates n_max to p_min
allocate(n_max, p_min);
A := A ∪ {n_max};
N := N \ {n_max};

; Allocates classifier to CPU
p_min := ARGMIN p IN P: start_time(classifier, p_min);
allocate(classifier, p_min);

; Returns the ready time of each CPU
RETURN {ready_time(CPU1), ..., ready_time(CPUP)};

```

---

**Fig. 4.** The processing time algorithm

same CPU. Otherwise, the communication time of the feature must be added to the feature receive time.

When allocating the non-allocated FE with the largest b-level to a CPU, the CPU which allows the earliest FE start time is selected (as a greedy step towards the goal of minimizing the estimated processing time of the processing DAGs). Consequently, the CPU selection is determined by the location of the FEs processing DAG parents and the communication time of the corresponding features.

This iteration continues until all the FEs have been allocated. Then the classifier is allocated to the CPU which minimizes the classifier start time. Finally, the scheduled stop time,  $ready\_time(CPU_i)$ , of the last FE to be executed on each CPU is returned. The estimated processing time of the media indexing application corresponds to the largest CPU ready time. The complete set of CPU ready times will be used to evaluate the processing characteristics of given processing DAGs in the next section.

## 6 The ARCAMIDE Algorithm

The *processing time* of a media indexing application depends on the DPE, the selection of FEs, the classifier as well as the scheduling of the FEs and the classifier on CPUs in

the DPE. The *error rate* of a media indexing application depends on the type of classifier used for indexing, the selection of FEs and the quality of the manually indexed media streams used for training. When the DPE, the scheduling algorithm, the classifier type and the manually indexed media streams are given, mainly the selection of FEs influences the error rate and processing time of the resulting media indexing application. In this section we propose a heuristic search algorithm, the ARCAMIDE algorithm, which selects sets of FEs from an FE library in order to offer tradeoffs between estimated error rate and processing time for a given media indexing application in a given DPE.

If the search for FE sets with appropriate tradeoffs between estimated error rate and processing time is to be computationally practical, only a very small number of the possible FE sets may be evaluated. When the FE library contains  $n$  FEs there may be up to  $2^n$  possible FE sets. Consequently, the choice of which FE sets to evaluate is of paramount importance.

The ARCAMIDE algorithm consists of two FE selection stages. In both stages of the algorithm the most inefficient parts of the processing DAGs in the FE library (when considering estimated error rate and processing time) are pruned. Roughly stated, this procedure corresponds to a standard sequential backward FE set search, extended to handle processing DAGs. The FE set search is performed backwards, rather than forwards, in order to avoid a computationally expensive  $n$ -step look-ahead search, made necessary by the processing DAGs and in some cases by complexly interacting features. Obviously, other FE set search procedures can be applied by the ARCAMIDE algorithm (such as beam search [8], genetic algorithms or branch and bound search [4]) by extending them to handle processing DAGs. However, due to its simplicity, computational efficiency and goal-directed behavior (when it comes to generating FE configurations in increasing order of estimated error rate and decreasing order of estimated processing time), this paper focuses on the sequential backward FE set search procedure.

The ARCAMIDE algorithm (based on the backward FE set search procedure) is summarized in Fig. 5 and described below in more detail.

Let  $F$  be the set of available FEs from the FE library. In stage one of the ARCAMIDE algorithm the individual error rate,  $er(\{f\} \cup desc(f, F))$ , of each FE,  $f$ , and its processing DAG descendants in  $F$ ,  $desc(f, F)$ , is estimated as described in Sect. 2. FEs whose error rate exceeds a given threshold  $i$  are considered irrelevant in the context of the given media indexing application. Consequently, these FEs are not considered in the computationally more complex stage two.

Stage two of the ARCAMIDE algorithm starts with the resulting set of relevant FEs,  $F$ , and proceeds by removing FEs (and their processing DAG descendants) from  $F$  sequentially until all the FEs have been removed. First, the parallelizability of  $F$  is considered by calculating the standard deviation of the CPU ready times,  $std\_dev(pt(F))$ , as returned by the scheduling algorithm from Sect. 5. If the standard deviation exceeds a threshold  $c$ ,  $F$  is considered to be unsuitable for parallelization due to the length of critical processing DAG paths (the longest path from an entry node to an exit node). To reduce the length of critical paths, only the FEs in  $F$  with the largest  $s$ -level

$$\underset{f \in F}{\operatorname{argmax}} s\_level(f)$$

---

```

; The ARCAMIDE algorithm
arcamide(F, i, c)
; Stage 1
FOR EACH f IN F
; Removes f if irrelevant
IF er({f} ∪ desc(f, F)) > i THEN
F := F \ ({f} ∪ desc(f, F));

; Stage 2
WHILE #F > 0 DO
; Determines whether critical paths
; need to be shortened
IF std_dev(pt(F)) > c THEN
; Shortens critical paths
F' := ARGMAX f IN F: s_level(f);
f_max := ARGMAX f IN F': ef(f, F);
ELSE
f_max := ARGMAX f IN F: ef(f, F);

; Removes f_max and its
; processing DAG descendants
F := F \ ({f_max} ∪ desc(f_max, F));

; Outputs F, error rate, and
; processing time of F
OUTPUT <F, er(F), max(pt(F))>;

RETURN;

```

---

**Fig. 5.** The ARCAMIDE algorithm

are considered for removal. On the other hand, if the standard deviation is less than or equal to  $c$ , all FEs in  $F$  are considered for removal.

Among the FEs considered for removal, the FE  $f$  that maximizes the efficiency function

$$ef(f, F) \equiv \frac{er(F) - er(F \setminus (\{f\} \cup desc(f, F)))}{max(pt(\{f\} \cup desc(f, F)))}$$

is removed from  $F$  in addition to its processing DAG descendants in  $F$ ,  $desc(f, F)$ . Here,  $er(S)$  denotes the estimated error rate of FE set  $S$ , and  $max(pt(S))$  denotes the estimated processing time of FE set  $S$ . In short, the efficiency function rewards FEs which contribute to maintaining the estimated error rate of  $F$  and which in addition are computationally cheap.

When the ARCAMIDE algorithm stops, it has generated a sequence of processing DAGs with different estimated error rate/processing time tradeoffs. The most appropriate tradeoff can then be selected based on the media indexing application at hand.

## 7 Empirical Results

In our media indexing application case from Sect. 2 the goal is to detect when a vehicle passes from the left to the right and when a vehicle passes from the right to the left. When manually selecting FEs for such a media indexing application, both the application specific indexing and the DPE specific processing characteristics of each FE must be considered. In this section we first discuss and model the indexing and processing characteristics of the FEs from Sect. 2. Then we discuss manual FE selection strategies. Finally, we compare these strategies with the sequence of processing DAGs produced by the ARCAMIDE algorithm when simulating the media indexing application case with respect to three different DPEs (1 CPU, 10 CPUs, 100 CPUs).

There are four different types of FEs related to each image region: motion FEs, difference picture FEs, texture FEs and colour FEs. These have different indexing and processing characteristics. In our model, when using a motion FE to detect a vehicle in an image region, the probability of false positives and false negatives are assumed to be 0.3. Likewise, when using a texture FE the probability of false positives is assumed to be 0.5 and the probability of false negatives is assumed to be 0.3. When using a colour FE the latter probabilities are reversed. A difference picture FE only produces intermediate results used by a motion FE. Obviously, the specified probabilities depend on the camera environment (e.g. sunshine, darkness, rain) and are here set to reflect rather difficult environment conditions. Finally, the processing time of each FE and the communication time of features between CPUs are as specified in Sect. 3. Note that the processing time of the classifier is not considered.

When manually selecting FEs the application specific indexing characteristics of each FE must be considered in addition to the processing characteristics. Firstly, FEs irrelevant to the indexing task need not be considered for inclusion in the processing DAGs. For instance, due to the chosen camera position in our media indexing application case, all passing vehicles are contained in the two middle image region rows. Accordingly, FEs located in the other image regions are irrelevant. The remaining FEs are candidates for inclusion. Secondly, when the goal is to differentiate between vehicles passing in the two different directions, FEs near the two middle columns are less accurate compared to FEs near the first and eighth column. This is because the passing direction determines when the vehicle will be detected in each region of the image and the difference between the left and right passing direction is most significant near the first and eighth column. For instance, irrespective of passing direction the vehicles will pass the middle regions at on average the same time. When it comes to processing characteristics, the FEs in an image region have similar efficiency, that is, the motion FEs produce more accurate vehicle detectors than colour and texture FEs, but have a comparable higher processing time. On the other hand, the texture and colour FEs in an image region can be executed in parallel, whereas the motion FE and the difference

picture FE must be executed in sequence. Consequently, texture FEs and colour FEs should be preferred before motion FEs when CPUs are abundant.

To evaluate the ARCAMIDE algorithm we trained two 10-state Hidden Markov Models (one for each vehicle passing direction) on 1000 simulated image sequences of vehicles moving from the left to the right and 1000 simulated image sequences of vehicles moving from the right to the left. Here, a vehicle appears on average twice in each image region of the two center rows when passing the camera view. We used another independent set of simulated image sequences (of identical size) to prune the processing DAGs. Similar, although more noisy, results were obtained with smaller sets of 100 and 500 simulated image sequences.

Given the simulated image sequences, the ARCAMIDE algorithm mimics the discussed FE selection strategies as follows. Firstly, in stage one of the ARCAMIDE algorithm, the FEs not located in the two middle image region rows were removed in addition to the FEs in the four center image regions. Here, the relevance threshold  $i$  was set to estimated error rate 0.45. Generally,  $i$  should reflect the number of concepts; e.g.  $i$  could be set to

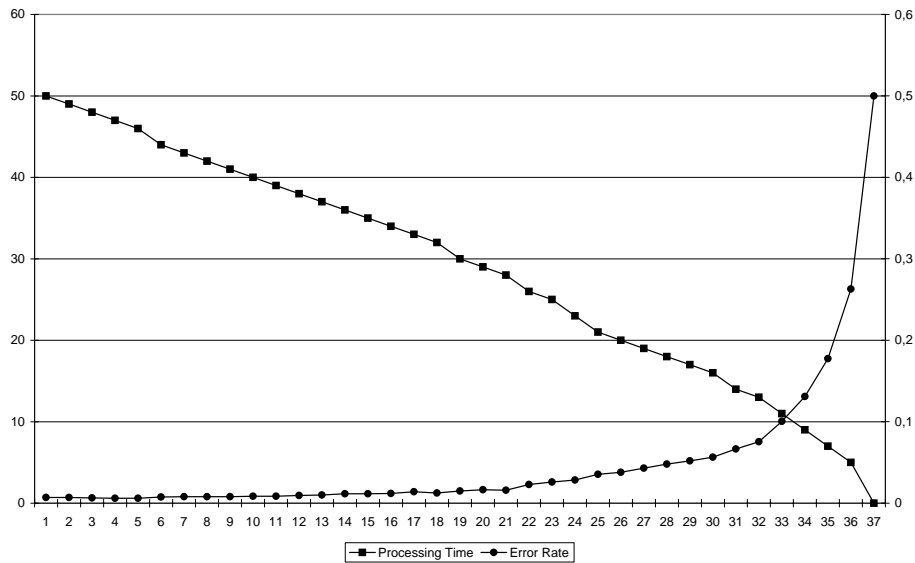
$$0.9(1 - \frac{1}{number\_of\_concepts}).$$

Secondly, in stage two of the ARCAMIDE algorithm, when trading off estimated error rate and processing time, the algorithm behaves as follows for each of the three different DPEs. When selecting FEs for a DPE consisting of 1 CPU, the ARCAMIDE algorithm do not consider the parallizeability of the processing DAGs, only the estimated FE error rate and processing time are considered. This can be seen from the ARCAMIDE FE removal order shown in Table 1. The corresponding estimated indexing error rates

**Table 1.** The FE removal order when the DPE consists of 1 CPU. In each table cell the difference picture FE, motion FE, texture FE and colour FE are listed from left-to-right top-to-bottom

36.	36.	31.	31.	6.	6.			22.	22.	5.	3.	25.	25.
20.	32.	21.	28.	2.	15.			10.	11.	16.	27.	4.	29.
33.	33.	34.	34.	19.	19.			24.	24.	35.	35.	37.	37.
17.	8.	13.	18.	12.	9.			1.	14.	23.	26.	7.	30.

and estimated processing times are shown in Fig. 6. As seen in Table 1, motion FEs seem to be slightly preferred before colour and texture FEs. Furthermore, FEs near the first and eighth image region columns are preferred before FEs near the middle image region columns. Consequently, as seen in Fig. 6, the estimated processing time can initially be reduced with little increase in estimated error rate (while inaccurate



**Fig. 6.** The estimated processing time and indexing error rate (y-axis) after each removal (x-axis) when the DPE consists of 1 CPU

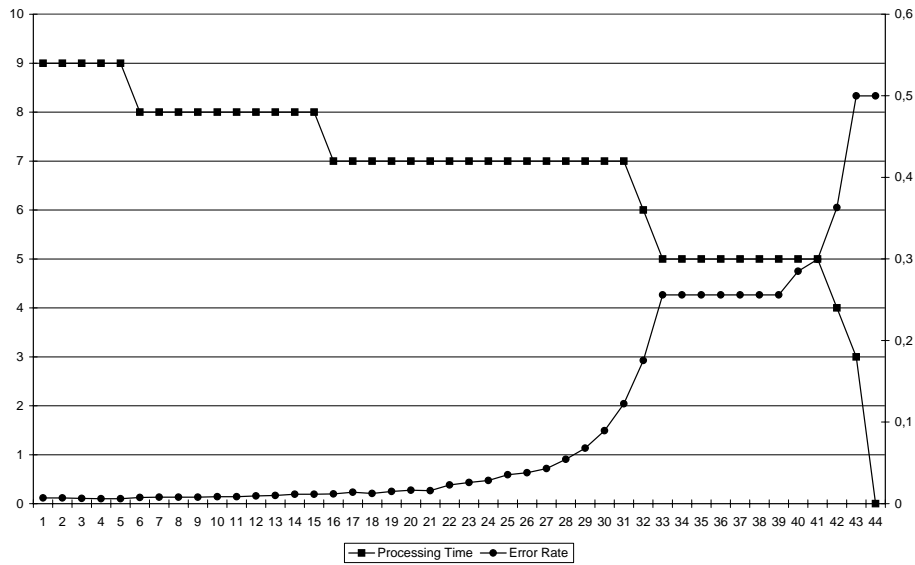
FEs are removed). However, later in the FE removal sequence, when FEs near the first and eighth image region columns must be removed, the estimated indexing error rate increases more dramatically.

When introducing 10 CPUs, the above removal strategy is followed by the AR-CAMIDE algorithm as long as the scheduling algorithm is able to uniformly distribute the FEs on the 10 CPUs. However, when the number of remaining FEs approaches the number of CPUs, the CPUs executing texture or colour FEs becomes on average ready 1 millisecond before CPUs executing motion and difference picture FEs. As a result, only the removal of motion FEs will reduce the processing time any further. The AR-CAMIDE algorithm detects this state by measuring the standard deviation of the CPU ready times, and changes removal strategy when the standard deviation exceeds a given threshold, in this case 1 (generally the threshold should be set to reflect the acceptable variation in CPU ready times). This can be seen in Table 2 and Fig. 7 from removal number 28 and thereafter.

When considering 100 CPUs, initially only the removal of motion FEs will reduce the processing time due to the dependency of motion FEs on picture difference FEs. As seen in Fig. 8 there are mainly two interesting processing DAGs; one containing all the relevant FEs and one containing only texture and colour FEs. The corresponding removal order is shown in Table 3.

**Table 2.** The FE removal order when the DPE consists of 10 CPUs. In each table cell the difference picture FE, motion FE, texture FE and colour FE are listed from left-to-right top-to-bottom

34.	31.	36.	28.	6.	6.			22.	22.	5.	3.	25.	25.
20.	42.	21.	40.	2.	15.			10.	11.	16.	27.	4.	43.
35.	32.	37.	29.	19.	19.			24.	24.	38.	30.	39.	33.
17.	8.	13.	18.	12.	9.			1.	14.	23.	26.	7.	41.



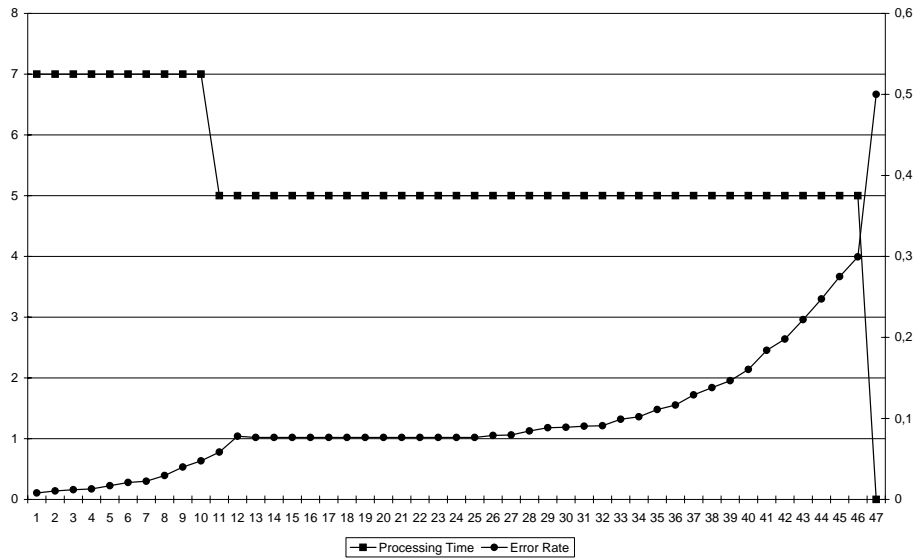
**Fig. 7.** The estimated processing time and indexing error rate (y-axis) after each removal (x-axis) when the DPE consists of 10 CPUs

## 8 Conclusion and Further Work

We have presented an algorithm for automatic resource aware construction of media indexing applications. The algorithm produces a sequence of processing DAGs which manifest various tradeoffs between estimated error rate and processing time, targeting a given DPE. The empirical results indicate that the algorithm are capable of producing useful media indexing application configurations for a moderate media indexing problem.

**Table 3.** The FE removal order when the DPE consists of 100 CPUs. In each table cell the difference picture FE, motion FE, texture FE and colour FE are listed from left-to-right top-to-bottom

14.	12.	16.	2.	18.	7.			20.	5.	22.	9.	24.	4.
37.	47.	42.	38.	28.	32.			30.	34.	41.	43.	40.	47.
15.	1.	17.	3.	19.	6.			21.	8.	23.	10.	25.	11.
44.	29.	39.	33.	13.	26.			27.	35.	31.	36.	46.	45.



**Fig. 8.** The estimated processing time and indexing error rate (y-axis) after each removal (x-axis) when the DPE consists of 100 CPUs

In our further work we will examine how adaptive media indexing applications can be constructed based on the produced sequence of processing DAGs. When more resources becomes available, the media indexing application can reconfigure to configurations earlier in the FE removal order, and when resources becomes unavailable, the media indexing application can reconfigure to configurations later in the FE removal order (i.e. graceful degradation of performance).

We will also conduct large scale experiments by using the ARCAMIDE algorithm to construct a traffic surveillance application consisting of several video cameras monitoring a road network where the concepts of interests may span several cameras.

In [11] a particle filter based approach for online hypothesis driven feature extraction is proposed. We will examine how this approach can be integrated in an online version of the ARCAMIDE algorithm.

Finally, we will evaluate the ARCAMIDE algorithm in various types of DPEs with the goal of refining the processing characteristics model, the DPE model and the processing time algorithm further.

## References

1. Hongeng, S., Bremond, F., Nevatia, R.: Bayesian Framework for Video Surveillance Applications. In: 15th International Conference on Pattern Recognition. Volume 1., IEEE (2000) 164–170
2. Eickeler, S., Muller, S.: Content-based Video Indexing of TV Broadcast News Using Hidden Markov Models. In: Conference on Acoustics, Speech and Signal Processing. Volume 6., IEEE (1999) 2997–3000
3. A. Pinz, M. Prantl, H.G., Borotschnig, H.: Active fusion—a new method applied to remote sensing image interpretation. Special Issue on Soft Computing in Remote Sensing Data Analysis **17** (1996) 1340–1359
4. Dash, M., Liu, H.: Feature Selection for Classification. Intelligent Data Analysis **1** (1997)
5. Eide, V.S.W., Eliassen, F., Lysne, O.: Supporting Distributed Processing of Time-based Media Streams. In: Distributed Objects and Applications, DOA2001. (2001) 281–288
6. Nakamura, Y., Nagao, M.: Parallel Feature Extraction System with Multi Agents -PAFE-. In: 11th IAPR International Conference on Pattern Recognition. Volume 2., IEEE (1992) 371–375
7. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley-Interscience. John Wiley and Sons, Inc. (2000)
8. Mitchell, T.M.: Machine Learning. Computer Science Series. McGraw-Hill International Editions (1997)
9. Pratt, W.K.: Digital Image Processing. Wiley Interscience. John Wiley & Sons, Inc. (1991)
10. Kwok, Y.K., Ahmad, I.: Benchmarking and comparison of the task graph scheduling algorithms. Journal of Parallel and Distributed Computing **59** (1999) 381–422
11. Granmo, O.C., Eliassen, F., Lysne, O.: Dynamic Object-oriented Bayesian Networks for Flexible Resource-aware Content-based Indexing of Media Streams. Scandinavian Conference on Image Analysis, SCIA'2001 (2001)