

Techniques for Parallel Execution of the Particle Filter

Ole-Christoffer Granmo¹, Frank Eliassen², Olav Lysne², and Viktor S. Wold Eide² *

¹ Department of Information and Communication Technology
Agder University College
Grooseveien 36, 4876 Grimstad, Norway
ole.granmo@hia.no
² Simula Research Laboratory
P.O. Box 134, 1325 Lysaker, Norway
{frank, olavly, viktore}@simula.no

Abstract. Dynamic Bayesian networks are a promising approach to automatic video content analysis which allows statistical inference and learning to be combined with domain knowledge. When the particle filter (PF) is used for approximate inference, video data can often be classified in *real-time* (supporting e.g. on-line automatic video surveillance). Unfortunately, the limited processing resources available on a typical host restricts the complexity, accuracy, and frame rate of PF classification tasks. Here, we target this limitation by applying the traditional parallel *pooled* classifiers architecture to execute multiple PFs in parallel and to coordinate their output. We then identify a significant weakness of this approach in terms of loss of accuracy. To reduce the loss of accuracy, we propose a novel scheme for coordinating the pooled PFs based on the exchange of so-called particles. In an object tracking experiment, a significant loss of accuracy is observed for the naive application of the pooled classifiers architecture. No loss of accuracy is detected when our scheme for exchanging particles is used.

1 Introduction

The technical ability to generate volumes of digital video data is becoming increasingly “main stream”. To utilize the growing number of video sources, both the ease of use and the computational flexibility of methods for content-based access must be addressed.

In order to make video content more accessible, pattern classification systems which automatically classify video content in terms of high-level concepts have been taken into use. The goal of such pattern classification systems is to bridge the gap between the low-level features produced through signal processing (e.g. color histograms or motion vectors) and the high-level concepts desired by the end-user (e.g. “running person”).

In this context, dynamic Bayesian networks (DBNs) [1] represent a particularly flexible class of pattern classifiers that allows statistical inference and learning to be combined with domain knowledge. Indeed, DBNs have been successfully applied to a wide range of video content analysis problems (e.g., [2, 3]). The successful application of DBNs can be explained by their firm foundation in probability theory, combined with the effective techniques for inference and learning that have been developed.

* This research is supported by the Norwegian Research Council under grant no. 126103/431.

The particle filter (PF) [4] is an approximate inference technique that opens up for *real-time* DBN-based video content analysis; i.e., video data can be classified on-line as video frames are captured. A real-time building surveillance system could for instance automatically classify whether someone is running rather than walking, on-line.

Unfortunately, the limited processing resources available on a typical host restricts the complexity, accuracy, and frame rate of PF classification tasks. E.g., rough tracking of the position of a single person in a single low rate video stream may be possible using a single CPU, but accurately tracking the position of multiple people as well as their interactions (talking, shaking hands, etc.) could require several CPUs. Obviously, coordinated analysis of multiple video streams (e.g. for building surveillance purposes) increases this processing bottleneck problem even further.

In this paper we target the classification processing bottleneck discussed above. In short, we propose techniques for parallel execution of the PF on multiple CPUs connected in a Local Area Network (LAN). In Sect. 2 we overview DBNs and PFs, which provide the basis for our work. Then, in Sect. 3 we show how the traditional parallel *pooled* classifiers architecture [5] can be used for executing multiple PFs in parallel and to coordinate their output. We also identify a significant weakness of this approach in terms of loss of accuracy. To reduce this loss we propose a novel scheme for coordinating the PFs, based on the exchange of so-called particles. The resulting techniques are evaluated empirically in Sect. 4. We conclude in Sect. 5.

2 Dynamic Bayesian Networks and Particle Filtering

We use DBNs to specify content analysis tasks, and we shall here give a short introduction. For a more thorough treatment, readers are referred to the literature (e.g. [1]). We also review the principles behind PF based inference for DBNs. Note that PFs, as well as the parallelization techniques we propose in this paper, do not need to be based on DBNs. Other classes of probabilistic models may also be used. However, we will here use DBNs for example purposes because of their modeling flexibility.

2.1 Dynamic Bayesian Networks (DBNs)

A DBN consists of a qualitative and a quantitative part. The qualitative part is a directed acyclic graph (DAG). The *nodes* in the DAG are variables with states. Each variable represent an entity of interest, and the states of a variable represent the states the corresponding entity can be in. For instance, consider a video image parted into 8×8 image blocks. A motion feature is extracted from each image block based on two consecutive video images, and the goal is to determine the coordinates of a tracked object from these motion features. Then, one DBN variable could represent the center block of the object. This variable would have 64 states, one for each block position. Furthermore, we could assign a motion feature variable to each image block with states “low”, “medium”, and “high”. These variables would represent the output of the motion feature extractors. The *directed links* in the DAG represent causal impact between the variables. E.g., a moving object influences the features output by the motion feature extractors, and therefore we should add a link from the object position variable to each motion feature variable.

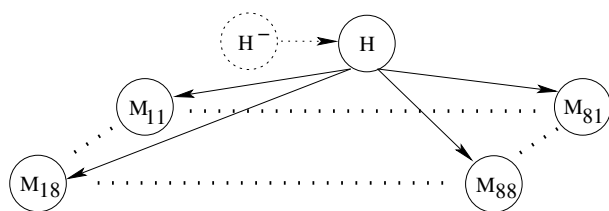


Fig. 1. A DBN modeling an object tracking content analysis task

So far, the above description corresponds to an ordinary Bayesian network (BN). The difference between a BN and a DBN is that in a DBN the DAG is sectioned into a sequence of identical *time slices*. Each time slice represents a particular point or interval in time. This means that links between variables in two consecutive time slices indicate a causal impact from time interval to time interval. Consequently, BNs are suited for modeling the content of an image, whereas DBNs are suited for modeling the content of a video stream. In a DBN we could for instance let each time slice correspond to a video stream image and then relate the possible positions of a moving object from time slice to time slice.

The complete DBN model suggested above represent an object tracking task and is illustrated in Fig. 1. The DBN in the figure consists of 8×8 motion feature variables M_{ij} , organized spatially as a grid, and a hypothesis variable H representing the block position of the tracked object. The inter time slice link (H^-, H) reflects that the position of the object depends on its position in the previous time slice. The link from H to a motion feature M_{ij} reflects the causal impact of each object position on the motion feature at grid position i, j . Note that in a stationary model, the causal impact between consecutive time slices does not vary. Accordingly, only the causal impact between two arbitrary consecutive time slices needs to be specified, as shown in the figure.

We now turn to the quantitative part of a DBN. In short, the strength of the directed links are represented as conditional probabilities. For each variable A with parents $pa(A)$, we specify the conditional probabilities $P(A|pa(A))$. If $pa(A) = \emptyset$ we specify the prior probability distribution $P(A)$. For the network in Fig. 1 we have to specify $P(H^-)$, $P(H|H^-)$, and $P(M_{11}|H) \cdots P(M_{88}|H)$; e.g., $P(M_{11} = high|H = [3, 3]) = 0.05$.

The above DBN model is used as a basis for the object tracking application described in [6]. Other kinds of content analysis tasks can be modeled in a similar manner. Before we turn to PFs and how they can be used for inference in DBNs, we briefly state the goal of our targeted kind of inference. In short, we target the calculation of *posterior probabilities*, that is, the calculation of the probability of certain DBN variables being in certain states, given the observed states of other DBN variables. For instance, assume that we only consider a single image and that the states of the motion feature variables have been observed for that image: $M_{11} = m_{11}, \dots, M_{88} = m_{88}$. We would then want to calculate $P(H = [i, j] | M_{11} = m_{11}, \dots, M_{88} = m_{88})$ for each coordinate $[i, j]$. This is because we want to identify the most probable object position for tracking purposes.

2.2 The Particle Filter (PF) and Inference in DBNs

There exist many algorithms for calculation of posterior probabilities in DBNs [1, 4]. The PF [4] is a *real-time* approximate technique. We describe the PF in two stages. I.e., we first describe the state of a PF at time slice t (including $t = 0$). From this state, posterior probabilities can be calculated given features observed up to and including time slice t . We then describe the procedure for advancing to time slice $t + 1$.

Let \mathcal{X}^t denote the set of DBN variables in time slice t . Furthermore, let \mathcal{H}^t denote the so-called hypothesis variables of time slice t , that is, the variables $\mathcal{H}^t \subseteq \mathcal{X}^t$ which cannot be observed directly. Finally, let \mathcal{F}^t denote the so-called feature variables in time slice t , i.e., the variables $\mathcal{F}^t \subseteq \mathcal{X}^t$ whose states are observed.

For the current time slice (t) our PF maintains a set S of *particles*. The particles can be seen as weighted samples. A single particle s consists of two parts. The first part of s is simply an assignment of a state x^t to each hypothesis variable $X^t \in \mathcal{H}^t$: $s.X^t := x^t$. Similarly, each feature variable $X^t \in \mathcal{F}^t$ is assigned its observed state. In our example, a particle assigns an object position to the current video image as well as e.g. a degree of motion to each video image block. The second part of s is a weight $s.w$ which determines the probability of the feature observations up to and including time slice t , given the states s assigns to the hypothesis variables: $P(\mathcal{F}^1 = f^1, \dots, \mathcal{F}^t = f^t | \mathcal{H}^1 = s.\mathcal{H}^1, \dots, \mathcal{H}^t = s.\mathcal{H}^t)$. Accordingly, this part indicates how accurately the assignments of s reflect the “truth”. By normalizing the weights of the particles, the particles can be seen as an approximation of the joint posterior probability distribution $P(\mathcal{H}^t | \mathcal{F}^1 = f^1, \dots, \mathcal{F}^t = f^t)$ [4]. From this joint probability distribution, the posterior distribution of individual hypothesis variables can be calculated. Note that each particle is initialized by sampling from a prior distribution $P(\mathcal{X}^0)$ specified as part of a DBN, e.g. $P(H^-)$ in our example DBN. The particle weights are set to 1 - no observations done for $t = 0$.

The question then is how to update the particle set S so that it covers a new time slice $t + 1$. An algorithm which performs this updating step is found in Fig. 2. Each particle s is updated as follows. First, s is replaced by a particle drawn randomly from the particle set. The particle is drawn according to the probability distribution derived by normalizing the particle weights, hereafter denoted $P(S')$. Generally, the resulting replaced particle set will provide a better starting point for approximating $P(\mathcal{H}^{t+1} | \mathcal{F}^1 = f^1, \dots, \mathcal{F}^{t+1} = f^{t+1})$ compared to the original particle set. This is because particles representing less likely scenarios are equivalently less likely to be included in the new particle set. Thus, the particles are concentrated on the likely scenarios and will not spread out across an exponential large set of possible, but very unlikely scenarios. As a second step, the DBN variables in time slice $t + 1$ are ordered topologically with respect to the DBN DAG and assigned states by particle s in that order. Each hypothesis variables $X^{t+1} \in \mathcal{H}^{t+1}$ is assigned a state drawn from the conditional probability distribution $P(X^{t+1} | pa(X^{t+1}) = s.pa(X^{t+1}))$. Because of the ordering of the variables, the particle has already assigned states to the parents $pa(X^{t+1})$ of X^{t+1} . The feature variables are treated differently as the states of these are given. As a result, the particle weight $s.w$ is updated to include each new observation ($s.X^{t+1} := \text{OBSERVE } X^{t+1}$): $s.w := s.w \times P(X^{t+1} = s.X^{t+1} | pa(X^{t+1}) = s.pa(X^{t+1}))$.

In combination, the above mechanisms evolve the particle set to be a rich summarization of likely video image content interpretations.

```

1: % Extend particles in  $S$  to cover time slice  $t+1$  from  $t$ 
2:  $S' := \text{copy}(S)$ ;
3: FOR EACH  $s \in S$  DO
4:   % Replace particle  $s$  based on sampling from
5:   % distribution defined by particle weights
6:    $s := \text{SAMPLE } P(S')$ ;
7:   % Assign state to DBN variables in new time slice
8:   FOR EACH  $X^{t+1} \in \mathcal{X}^{t+1}$  DO
9:     IF  $X^{t+1} \in \mathcal{H}^{t+1}$  THEN
10:       $s.X^{t+1} := \text{SAMPLE } P(X^{t+1} | pa(X^{t+1}) = s.pa(X^{t+1}))$ ;
11:     ELSE
12:       $s.X^{t+1} := \text{OBSERVE } X^{t+1}$ ;
13:       $s.w := s.w \times P(X^{t+1} = s.X^{t+1} | pa(X^{t+1}) = s.pa(X^{t+1}))$ ;

```

Fig. 2. The updating step of the PF algorithm when applied to DBNs

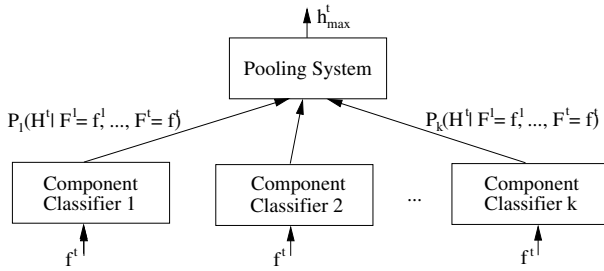


Fig. 3. The pooled classifiers architecture applied to PFs

3 Techniques for Parallel Execution of the Particle Filter

The limited processing resources available on a typical host restricts the complexity, accuracy, and frame rate of PF classification tasks. In this section we describe the *pooled* classifiers architecture [5] and propose how this architecture can form the basis for a parallel multi-PF. In essence, particles are processed in parallel on multiple CPUs to resolve the PF classification processing bottleneck. We also propose a communication scheme for exchanging particles, with the goal of increasing the classification accuracy.

3.1 Applying the Pooled Classifiers Architecture to PFs

The traditional pooled classifiers architecture is shown in Fig. 3. k independent classifiers take a set of features as input. The classifiers are executed in parallel so that k classifications are output. A pooling system aggregates these outputs to make a final classification.

We use PFs as classifiers. Each PF i takes the features extracted in the current time slice t as input, i.e. $\mathcal{F}^t = f^t$. Based on this input a PF i is able to output an approximation of $P(\mathcal{H}^t | F^1 = f^1, \dots, F^t = f^t)$, as we observed in the previous section. Alternatively, the joint distribution $P(\mathcal{H}^t, F^1 = f^1, F^t = f^t)$ could be given as output. These variants are hereafter referred to as *TI* and *III* respectively. In turn, the pooling system sums and normalize the resulting local approximations, to produce a global posterior probability distribution. This global probability distribution is used to identify the a posteriori most probable state $\mathcal{H}^t = h_{\max}^t$, which is given as output from the pooling system. The overall process is illustrated in Fig. 3.

Unfortunately, the above approach to parallelizing a PF has a significant disadvantage. In effect, the approach fragments the particles of the PF to be parallelized into k small sets, one for each classifier component. This influences the particle replacement conducted at line 6 of the PF algorithm in Fig. 2. To explain the consequences of this fact, let us first consider some of the characteristics of the traditional PF. The traditional PF is mainly model-driven and not data-driven. That is, only the prior hypothesis state distribution of a new time slice t , as modeled by the DBN, is used for sampling (see line 10 from Fig. 2). The features extracted in the time slice (the data) are not considered at all when sampling the hypothesis states. So, if the true states of the hypothesis variables are improbable given the prior distribution, few particles will match those states (how few depends on how improbable the true states are). For instance, assume that a DBN models the movement patterns of a certain class of airplanes. If the pilot of an airplane that is tracked by a PF manages to make a difficult and therefore improbable maneuver, few particles will match that move. This may cause problems for the PF. Indeed, if no particles match the true hypothesis states, the PF may lose track of the current real-world situation. On the other hand, if even a single particle is able to match the true hypothesis states, the particle replacement step of line 6 in Fig. 2 makes sure that most of the particles are brought back on track.

However, this particle replacement effect is significantly reduced when the particles are fragmented into small sets as in the above pooled classifiers architecture: *particles are only replaced from the locally available particles*. Consequently, each individual PF in the pool may lose track of the real-world situation one-by-one. As PFs start losing track of the real-world situation, increasingly amounts of noise are introduced to the pooling system. Consequently, the classification suffers.

3.2 A Scheme for Exchanging Particles

To overcome the above fragmentation problem, we now introduce a scheme for exchanging particles between the pooled PFs. The main purpose of exchanging particles is to bring strayed PFs back on track.

The scheme takes advantage of LAN support for broadcasting messages, allowing messages to be transferred between machines in a few microseconds (an insignificant amount of time in the context of video content analysis). Note that other classes of efficient one-to-many communication mechanisms may also be used, such as multicast, as seen in [6]. Broadcasting a particle means that the particle is received by all the pooled PFs by the means of a single LAN transmission. In contrast, traditional UDP/TCP communication (unicast) requires one LAN transmission for each recipient. Accordingly, if

each of the k PFs in the pool are to submit a particle to each of the other PFs in the pool, we would need $k(k - 1)$ LAN transmissions for unicast. By using broadcast, on the other hand, we only need k LAN transmissions for this particular particle exchange. Accordingly, by taking advantage of the broadcast facilities of a LAN, efficient sharing of particles may be achieved.

To elaborate, we add a particle broadcast step to the PF algorithm from Fig. 2:

```
14: BROADCAST  $\operatorname{argmax}_{s \in S} s.w$ ;
```

In this step, each PF identifies the particle with the largest weight and then broadcasts this particle to the other PFs in the pool. This means that when each PF advances to a new time slice, they have been supplied with $k - 1$ additional particles. As a consequence, line 2 of the PF algorithm is modified so that local particles also can be replaced by the supplied particles (S_B):

```
2:  $S' := \operatorname{copy}(S \cup S_B)$ ;
```

Note that the particle weights have to be normalized repeatedly in order to avoid the difficulty of representing small real-valued numbers on computers. In our distributed scheme, normalization is supported by piggy-backing normalization constants with the broadcasted messages.

To conclude, the scheme makes sure that each PF is provided with the locally most likely particles in the PF pool, by only transmitting a number of messages equal to the number of PFs across the LAN at each time step. Most importantly, this scheme guarantees that the globally most likely particle in the PF pool always is shared between the PFs. In the following section we denote this technique as *TIII*.

4 Empirical Results

In this section we evaluate the proposed techniques empirically by the means of the DBN object tracking case. In order to challenge the model-driven PF approach, we simulate an object that makes unexpected/improbable moves from time slice to time slice. The tracked object could for instance be an airplane whose pilot tries to confuse the tracker by making unexpected/improbable maneuvers. Thus, we let the object movement between two time slices have probability 0.035 of occurring a priori. With 128 particles (on track) the probability that no particle matches a given movement is then $0.965^{128} \approx 0.01$, and with 16 particles the probability is $0.965^{16} \approx 0.57$.

The techniques *TI*, *TII*, and *TIII* are tested with 8 PFs, each maintaining 16 particles. Also, a traditional PF maintaining 128 particles is tested for comparison purposes. The number of time slices passed before a technique loses track of the object is counted, and the probability of keeping track of the object at each time slice is calculated based on 10000 simulation runs. The results are shown in Fig. 4. As seen in the figure, the results confirm our reasoning from the previous section. Also, the performance of *TIII* and the traditional PF is comparable. In fact, we were not able to discriminate them empirically.

We have also tested the scalability of the proposed techniques [6]. A real object tracking application has been implemented, and the processing rates achieved with respectively 1, 2, 4, and 5 pooled PFs were measured. The processing rates of the proposed techniques seem to increase linearly with the number of CPUs.

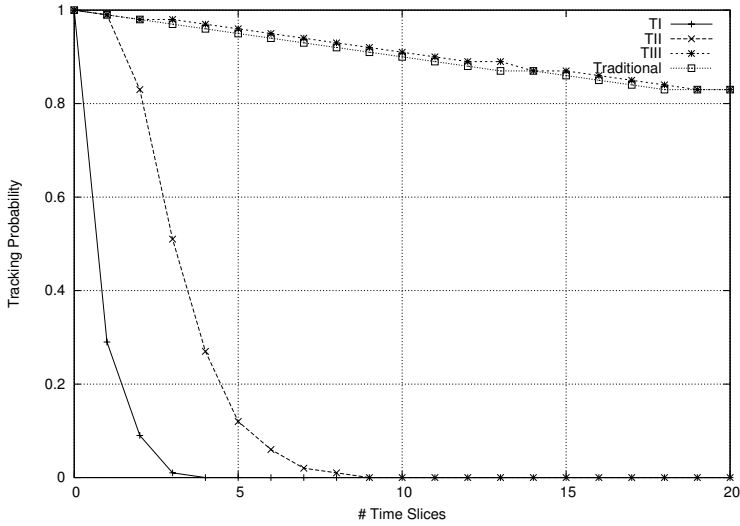


Fig. 4. The tracking probability after # time slices

5 Conclusion

The limited processing resources available on a typical host restricts the complexity, accuracy, and frame rate of PFs. Here, we have proposed techniques for parallel execution of the PF, based on the pooled classifiers architecture. A loss of accuracy caused by parallelization is avoided by allowing the pooled PFs to exchange particles. Equally important, the processing rates of the techniques seem to increase linearly with the number of CPUs, indicating a feasible solution to the targeted processing bottleneck problem.

References

1. Jensen, F.V.: Bayesian Networks and Decision Graphs. Series for Statistics for Engineering and Information Science. Springer Verlag (2001)
2. Chang, S.F., Sundaram, H.: Structural and Semantic Analysis of Video. In: Multimedia and Expo 2000 IEEE. Volume 2. (2000) 687–690
3. Naphade, M., Huang, T.: Extracting semantics from audio-visual content: the final frontier in multimedia retrieval. IEEE Transactions on Neural Networks (2002) 793–810
4. Liu, J.S., Chen, R.: Sequential Monte Carlo methods for Dynamic Systems. Journal of the American Statistical Association **93** (1998) 1032–1044
5. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley-Interscience. John Wiley and Sons, Inc. (2000)
6. Eide, V.S.W., Eliassen, F., Granmo, O.C., Lysne, O.: Scalable Independent Multi-level Distribution in Multimedia Content Analysis. In: Protocols and Systems for Interactive Distributed Multimedia (IDMS/PROMS 2002). Lecture Notes in Computer Science 2515, Springer (2002) 37–48