Decentralized algorithms for classifier topology optimization in large-scale multi-concept detection

ABSTRACT

Multi-concept identification in high volume multimedia streams is critical for a number of applications, including large-scale multimedia analysis, processing, and retrieval. Content of interest is filtered using a collection of binary classifiers that are deployed on distributed resource-constrained infrastructure. In this paper, we design distributed algorithms for determining the optimal topology of single concept detectors (classifiers) to identify the multiple concepts of interest. These algorithms dynamically order individual classifiers into chain topologies to tradeoff accuracy against processing delay, based on underlying data characteristics, system resource constraints as well as the performance and complexity characteristics of each classifier.

1. INTRODUCTION

There is a large class of emerging applications for annotation, knowledge extraction and online search and retrieval that require the analysis of high volume data streams in real time. Examples of these applications include online financial analysis, fraud detection, photo and video retrieval, spam classification, medical services, search engines, etc. Each application can be viewed as a processing pipeline that analyzes high volume data from a set of data sources to extract valuable information in real time. Furthermore, depending on the underlying data characteristics, volume, and concepts of interest, the mining tasks may be computationally very challenging. For instance, YouTube users currently upload more than 10 hours of video every minute. If we apply Support Vector Machine (SVM) models trained on 100,000 images on such data, it can take up to 270,000 2.16 GHz Dual-Core CPUs to detect 1000 concepts in real time.

The development of large-scale stream mining platforms has enabled applications to be constructed as topologies of distributed operators deployed on a set of heterogeneous processing resources. Constructing applications as topologies of distributed processing operators can lead to enhanced scalability, reliability and performance-complexity tradeoffs – allowing us to achieve the desired performance. Specifically, we build multi-concept detection applications using a topology of low-complexity binary classifiers, each performing feature extraction and classification specific to individual concepts of interest [6]. In this paper, we focus on constructing and ordering classifiers into chain topologies on distributed processing nodes tailored towards answering specific queries of interest on data streams, while trading off accuracy against processing delays.

This work lies at the intersection of research in large-scale stream mining, and the linear topology ordering problems. Prior research on stream mining optimization [1] is focused on resource allocation in resource-constrained topologies. More recent works [5] have concentrated on optimizing classification accuracy under resource constraints, yet without considering the classifier ordering problem: the topology of the classifiers are given *a-priori* and the optimization performed takes these topologies as granted. Furthermore, processing delay has seldom been studied [4] and always at equilibrium. Finally, there has been a significant body of work on using ensembles of classifiers to improve classification accuracy [7]. The ordering problem has been studied as part of the broader pipeline-ordering problem [2] as well as the classical set-cover problem [11]. These approaches design low-complexity algorithms to minimize the end-to-end processing time. However, they always consider perfect classifiers, i.e. classifiers that make no mistakes, and neglect resource constraints. In our case, we consider the general case of imperfect classifiers distributed on a resource-constrained network.

A centralized approach of stream mining ordering has also been studied in our paper [3]. In general, centralized approaches are incompatible with the distributed nature of the stream mining system, as they are limited in terms of scaling, adaptation to dynamics, and failure resilience. In this paper, we determine the optimal topology of a stream mining system using a decentralized approach. Our algorithms combines stochastic decision processes and reinforcement learning techniques.

This paper is organized as follows. In Section 2, we provide a background on classifiers used for concept detection and underline the accuracy tradeoff between detection and false alarm. In Section 3, we formulate the order and operating point selection as an optimization problem aiming to achieve the highest accuracy of classification with the least delay. Section 4 introduces a stochastic decision process where order and operating point are selected locally by each classifier in a decentralized fashion, using limited message exchange. In Section 5, we construct decentralized search algorithms, based on reinforcement learning techniques. Our results are presented in Section 6. Finally, in Section 7, we consider the case of resource-constrained stream mining, where the system gains in being organized as multiple parallel chains. Conclusions are drawn in Section 8.

2. BINARY CLASSIFICATION

Multimedia retrieval and analysis applications pose queries on data that require multiple concepts to be identified. More specifically, a query is answered as a conjunction of a set of N classifiers $\mathcal{C} = \{C_1, ..., C_N\}$, each associated with a concept to be identified. By partitioning the problem into this ensemble of classifiers and filtering data (i.e. discarding data

that is labeled "no" by any classifier) successively; we can control the amount of resources consumed by each classifier in the ensemble. This justifies using a chain topology of classifiers, where the output of one classifier is passed to another classifier, etc, as shown in Fig. 1.

Definition: **Order of a set of classifiers.** *An order of a set of classifiers $\{C_1, ...C_N\}$ is a permutation $\sigma \in \mathcal{P}erm(N)$ such that input data flows from $C_{\sigma(1)}$ to $C_{\sigma(N)}$.*

Throughout this paper, we will generically use the index i to identify a classifier and h to refer to its depth in the chain of classifiers. Hence, $C_i = C_{\sigma(h)}$ will mean that the h^{th} classifier in the chain is C_i.

A-priori selectivities Each binary classifier $C_i = C_{\sigma(h)}$ labels input data into two classes $\mathcal{H}_i$ (class of interest) and $\overline{\mathcal{H}_i}$. Data labeled as belonging to $\mathcal{H}$ is forwarded, while data labeled as belonging to $\overline{\mathcal{H}}$ is dropped. If input data into the classifier is represented as X, each classifier has a *a-priori* conditional selectivity $\phi_h^\sigma = \mathbb{P}(X \in \mathcal{H}_{\sigma(h)} | X \in \bigcap_{k=1}^{h-1} \mathcal{H}_{\sigma(k)})$ corresponding to the conditional probability of data belonging to classifier $C_i = C_{\sigma(h)}$'s class of interest, given that it belongs to the class of interest of the previous $h-1$ classifiers. Similarly, we define the negative *a-priori* selectivity as $\overline{\phi_h^\sigma} = \mathbb{P}(X \in \mathcal{H}_{\sigma(h)} | X \notin \bigcap_{k=1}^{h-1} \mathcal{H}_{\sigma(k)})$. Note that these *a-priori* selectivities are inherent to the data features and to the relationships between concepts; they do not depend on the operating point for individual classifiers.

Classifier performance. The performance of the classifier is characterized by its Detection Error Tradeoff (DET) curve that represents tradeoffs between probability of detection p^D and probability of false alarm p^F. We represent the DET curve as a function $f : p^F \mapsto p^D$ that is increasing, concave and lies over the first bisector. As a consequence, an operating point on this curve is parametrized uniquely by its false alarm ratios p^F.

We model the average time needed for a classifier to process a stream tuple as α (in seconds). The order of magnitude of α depends on the nature of the data, as well as the classification algorithm, and can vary from microseconds (screening text) to multiple seconds (complex image classification).

Throughput and goodput of a chain of classifiers. The forwarded output of a classifier consists of correctly labeled data from class $\mathcal{H}$ as well as false alarms from class $\overline{\mathcal{H}}$. We use g to represent the goodput (portion of data correctly labeled) and t to represent the throughput (total forwarded data, including mistakes). As in [5], we can derive the throughput t_i and goodput g_i of classifier $C_i = C_{\sigma(h)}$ recursively as

$$\begin{bmatrix} t_i \\ g_i \end{bmatrix} = \underbrace{\begin{bmatrix} p_i^F + \overline{\phi_h^\sigma}(p_i^D - p_i^F) & (\phi_h^\sigma - \overline{\phi_h^\sigma})(p_i^D - p_i^F) \\ 0 & \phi_h^\sigma p_i^D \end{bmatrix}}_{T_i = T_h^\sigma} \begin{bmatrix} t_{h-1}^\sigma \\ g_{h-1}^\sigma \end{bmatrix} \tag{1}$$

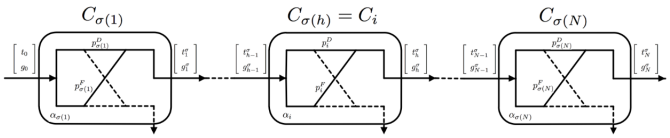

Figure 1: Classifier chain

3. PROBLEM FORMULATION

Misclassification cost. The misclassification cost, or error cost, may be computed in terms of the two types of accuracy errors – a penalty c^M per unit rate of missed detection, and a penalty c^F per unit rate of false alarm. These are specified by the application requirements. Noting $\Phi = \prod_{h=1}^N \phi_h^\sigma$, the total misclassification cost is

$$c_{err}^\sigma = c^M \underbrace{(\Phi t_0 - g_N^\sigma)}_{missed\ data} + c^F \underbrace{(t_N^\sigma - g_N^\sigma)}_{wrongly\ classified\ data} \tag{2}$$

Processing delay cost. Delay may be defined as the time required by the classifiers chain to process a stream tuple. Given that classifier $C_{\sigma(h)}$ processes a fraction $\frac{t_h^\sigma}{t_0}$ of data , the average end-to-end processing time required by the σ-ordered chain to process stream data is

$$c_{delay}^\sigma = \sum_{k=1}^{N} \alpha_{\sigma(k)} t_{k-1}^\sigma \tag{3}$$

Optimization Problem Analysis. The utility function may be defined as the negative weighted sum of both the misclassification cost and the processing delay cost: $U = -c_{err} - \lambda\, c_{delay}$, where the parameter λ controls the tradeoff between misclassification and delay. This utility is a function of the throughputs and goodputs of the stream within the chain, and therefore implicitly depends on:

1. The order σ in which the classifiers are arranged
2. The operating point $x_i \in [0,1]$ selected by each classifier , where $(p_i^F, p_i^D) = (x_i, f_i(x_i))$.

The optimization problem can be formulated as:

$$\begin{cases} \underset{\substack{\sigma \in \mathcal{P}erm(N) \\ \mathbf{x} \in \mathbb{R}^N}}{\text{maximize}} & U(\sigma, \mathbf{x}) = g_N - K t_N^\sigma - \sum_{k=1}^{N} \rho_{\sigma(k)} t_{k-1}^\sigma \\ \text{subject to} & 0 \le x \le 1 \end{cases} \tag{4}$$

where $K = \frac{c^F}{c^F + c^M} \in [0,1]$ and $\rho = \frac{\lambda}{c^F + c^M} \alpha \in {\mathbb{R}^+}^N$.

Requirement for algorithms to meet time constraints. Proposed algorithms must take into account system's dynamics, both in terms of stream characteristics evolutions and classifiers' processing time variations. This time-dependency is yet all the more true in a multi-query context, because the overall data stream processed is not homogeneous, since it is the concatenation of chunks of data stream from each query. Such plurality of queries can lead to very abrupt changes over small intervals of time: two consecutive tuples could belong to streams corresponding to two different queries, each with its specific selectivities, processing time and, most importantly, with its specific set of classifiers to go through.

These dynamics requires rapid adaptation of the order and operating points, often even at the granularity of one tuple. Any optimization algorithm thus needs to provide a solution with a time granularity finer than the system dynamics.

Denote by τ the amount of time required by an algorithm to perform one iteration, i.e to provide a solution to the order and configuration selection problem. The solution given by an algorithm will not be obsolete if

$$\tau \le \mathfrak{C} \tau^{dyn} \tag{5}$$

where τ^{dyn} represents the characteristic time of significant change in the input data and characteristics of the stream mining system and $\mathfrak{C} \le 1$ represents a buffer parameter in case of bursts.

Observe that centralized algorithm are not systematically compliant to this time requirement. Indeed, at the contrary of decentralized algorithms, centralized algorithms cannot trade-off performance and computational time, and therefore τ may be superior to dynamics.

4. DECENTRALIZED APPROACH

4.1 Decentralized optimization problem

The key idea of the decentralized algorithm is to use local decisions consisting in determining to which classifier to forward the stream. To describe this, we set up a stochastic decision framework $\{\mathcal{C}, \mathcal{S}, \mathcal{A}, \mathcal{U}\}$, illustrated in Fig. 2, where

- $\mathcal{C} = \{C_1, ..., C_N\}$ represents the set of classifiers.
- $\mathcal{S} = \underset{i \leq N}{\times} \mathcal{S}_i$ represents the set of states.
- $\mathcal{A} = \underset{i \leq N}{\times} \mathcal{A}_i$ represents the set of actions.
- $\mathcal{U} = \{U_1, ..., U_N\}$ represents the set of utilities.

Users of the stream mining system. Consider N classifiers $\mathcal{C} = \{C_1, ..., C_N\}$ with the notation $C_i = C_{\sigma(h)}$. The classifiers are supposed autonomous: they do not communicate with each other and take decisions independently. We will also refer to the stream source as $C_0 = C_{\sigma(0)}$.

States observed by each classifier. The local state set of classifier $C_i = C_{\sigma(h)}$ is $\mathcal{S}_i = \{(Children(C_i), \theta_i)\}$.

$Children(C_i) = \left\{C_k \in \mathcal{C} | C_k \notin \{C_{\sigma(1)}, C_{\sigma(2)} ..., C_i\}\right\}$ represents the subset of classifiers through which the stream still needs to be processed after it passes classifier C_i. This is a required identification information to be included in the overhead of each stream tuple in order to know to which classifier the stream should be forwarded, instead of sending it to a classifier which has already processed it.

The throughput-to-goodput ratio $\theta_i = \frac{t^\sigma_{h-1}}{g^\sigma_{h-1}} \in [1, \infty]$ is a measure of the accuracy of the ordered set of classifiers $\{C_{\sigma(1)}, C_{\sigma(2)}, ..., C_i\}$. Indeed, $\theta_i = 1$ corresponds to perfect classifiers $C_{\sigma(1)}, C_{\sigma(2)}, ..., C_i$, (with $p^D = 1$ and $p^F = 0$), while large θ_i imply that data has been either missed or wrongly classified. The state θ_i can be passed along from one classifier to the next in the stream tuple overhead.

Since $\theta_i \in [1, \infty]$, the set of states $\mathcal{S}_i$ is of infinite cardinality. For computational reasons, we would require a finite set of actions. We will therefore approximate the throughput-to-goodput ratio by partitioning $[1, \infty]$ into L bins $S_l = [b_{l-1}, b_l]$ and approximate $\theta_i \in S_l$ by some fixed value $s_l \in S_l$.

Actions of a classifier. Each classifier C_i has two independent actions: it selects its operating point x_i and it chooses among its children the classifier $C_{i\rightarrow}$ to which it will transmit the stream. Hence $\mathcal{A}_i = \{(x_i, C_{i\rightarrow})\}$, where

- $x_i \in [0, 1]$ corresponds to the operating point selected by C_i.
- $C_{i\rightarrow} \in Children(C_i)$ corresponds to the child classifier to which C_i will forward the stream. We will refer to $C_{i\rightarrow}$ as the trusted child of classifier C_i.

Note that the choice of trusted child $C_{i\rightarrow}$ is the local equivalent of the global order σ. The order is constructed classifier by classifier, each one selecting the child to which it will forward the stream: $\forall h \in [1, N], C_{\sigma(h)} = C_{\sigma(h-1)\rightarrow}$.

Local utility of a classifier. We define the local utility of a chain of classifiers using backward induction:

$$\begin{array}{rcl} U_{\sigma(N)} & = & -\rho_{\sigma(N)} t^\sigma_{N-1} + g^\sigma_N - K t^\sigma_N \\ U_{\sigma(h)} & = & -\rho_{\sigma(h)} t^\sigma_{h-1} + U_{\sigma(h+1)} \end{array} \quad (6)$$

The end-to-end utility of the chain of classifiers can then be reduced to $U = U_\sigma(1)$.

Proposition. *Each classifier $C_i = C_{\sigma(h)}$ will globally maximize the system's utility by autonomously maximizing its local utility $U_i = \left[\begin{array}{cc} v^\sigma_h & w^\sigma_h \end{array}\right] \left[\begin{array}{c} t^\sigma_{h-1} \\ g^\sigma_{h-1} \end{array}\right]$ where the local utility parameters $\left[\begin{array}{cc} v^\sigma_h & w^\sigma_h \end{array}\right]$ are defined recursively:*

$$\begin{array}{rcl} \left[\begin{array}{cc} v^\sigma_N & w^\sigma_N \end{array}\right] & = & -\left[\begin{array}{cc} \rho_{\sigma(N)} & 0 \end{array}\right] + \left[\begin{array}{cc} -K & 1 \end{array}\right] T^\sigma_N \\ \left[\begin{array}{cc} v^\sigma_h & w^\sigma_h \end{array}\right] & = & -\left[\begin{array}{cc} \rho_{\sigma(h)} & 0 \end{array}\right] + \left[\begin{array}{cc} v^\sigma_{h+1} & w^\sigma_{h+1} \end{array}\right] T^\sigma_h \end{array}$$

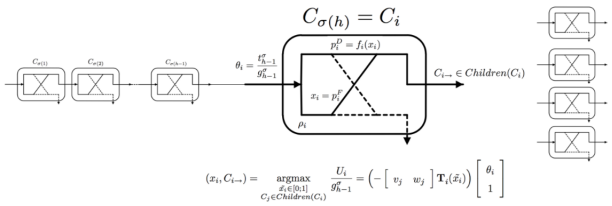

Figure 2: Stochastic decision process

Decision taking The local utility of classifier C_i can be rewritten as

$$U_i = \left(-\left[\begin{array}{cc} \rho_i & 0 \end{array}\right] + \left[\begin{array}{cc} v^\sigma_{h+1} & w^\sigma_{h+1} \end{array}\right] T_i(x_i)\right) \left[\begin{array}{c} t^\sigma_{h-1} \\ g^\sigma_{h-1} \end{array}\right] \quad (7)$$

Therefore, the decision of classifier C_i only depends on the state θ_i which it observes and on the local utility parameters $\left[\begin{array}{cc} v_j & w_j \end{array}\right]$ of its children classifiers $C_j \in Children(C_i)$. Being provided with the utility parameters of all his children, classifier C_i can then uniquely determine its best action, i.e, its operating point x_i and its trusted child in order to maximize its local utility.

4.2 Decentralized ordering algorithms

At this stage, we consider classifiers with fixed operating points. The action of a classifier C_i is therefore limited to selecting to which classifier $C_{i\rightarrow} \in Children(C_i)$ it will forward the stream.

4.2.1 Exhaustive Search Ordering Algorithm

We will say that a classifier C_i *probes* a child classifier C_j when it requests its child utility parameters $\left[\begin{array}{cc} v_j & w_j \end{array}\right]$.

To best determine its trusted child, a classifier only requires knowledge on the utility parameters of all its children. We can therefore build a recursive algorithm as follows: all classifiers are probed by the source classifier C_0 ; to compute their local utility, each of them then probes its children for their utility parameters $\left[\begin{array}{cc} v & w \end{array}\right]$. To determine these, each of the probed children needs to probe its own children for their utility parameter, etc. The local utilities are computed in backwards order, from leaf classifiers to the root classifier C_0, as shown in Fig. 3. The order yielding the maximal utility is selected.

Observe that this decentralized ordering algorithm leads to a full exploration of all $N!$ possible orders. Achieving the optimal order only requires one iteration, but this iteration however consists in $O(N!)$ operations and may thus require

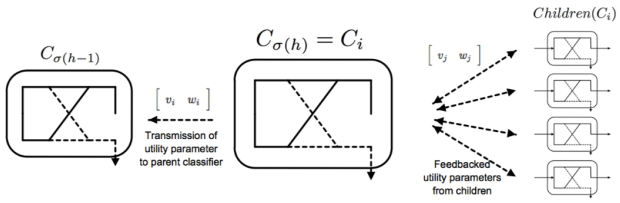

Figure 3: Information fed back

substantial time[1]. For quasi-stationary input data, the ordering could be performed offline and such computational time requirement would not affect the system's performance. However, in bursty and heterogeneous settings, we have to ensure that the order calculated by the algorithm would not arrive too late and thus be completely obsolete. In particular, the time constraint $\tau \leq \mathfrak{C}\tau^{dyn}$, underlined in Section 3, must not be violated.

We therefore need algorithms capable of quickly determining a good order, though convergence may require more than one iteration. In this way, it will be possible to reassess the order of classifiers on a regular time basis to adapt to the environment.

4.2.2 Global Partial Search Ordering Algorithm

The key insight of the Partial Search Algorithm is to screen only through a selected subset of the $N!$ orders at each iteration. Instead of probing all its children classifiers systematically, the h^{th} classifier will only request the utility parameters $\left[\begin{array}{cc} v & w \end{array}\right]$ of a subset of its $N-h$ children.

From a global point of view, one iteration can be decomposed in three major steps, as shown on Fig. 4:

Step 1 **Selection of the children to probe.** A partial tree is selected recursively (light grey on Fig. 4). A subset of the N classifiers are probe as first classifier of the chain. Then, each of them selects the children it wants to probe, each of these children select the children which it wants to probe, etc.

Step 2 **Determination of the trusted children.** The order to be chosen is determined backwards: utilities are computed from leaf classifiers to the source classifier C_0 based on feedbacked utility parameters. At each node of the tree, the child classifier which provides its parent with the greatest local utility is selected as trusted child (dark grey on Fig. 4).

Step 3 **Stream processing.** The stream is forwarded from one classifier to its trusted child (black on Fig. 4).

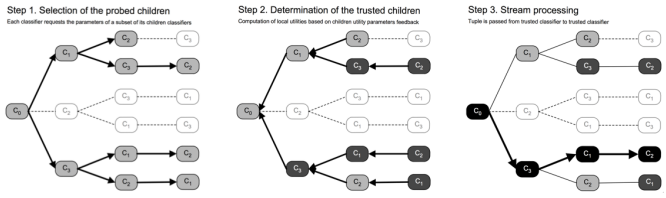

Figure 4: Global Partial Search Algorithm

4.2.3 Local Partial Search Ordering Algorithm

Observe that one classifier may appear at multiple depth and position in the classifiers' tree. Each time, it will realize the following local algorithm:

[1]This can take $\tau > 5$ minutes for 7 classifiers (see Section 6)

Algorithm 1 Partial Search Ordering Algorithm - *for classifier* $C_i = C_{\sigma(h)}$

1. **Observe state** $(\theta_i, Children(C_i))$
2. With probability p_j^i, **request utility parameters** $\left[\begin{array}{cc} v_{\sigma(h+1)} & w_{\sigma(h+1)} \end{array}\right] = \left[\begin{array}{cc} v_j & w_j \end{array}\right]$ for any of the $N-h$ classifiers $C_j \in Children(C_i)$
3. For each child probed, **compute** corresponding **utility** $U_i(C_j) = \left(-\left[\begin{array}{cc} \rho_{\sigma(i)} & 0 \end{array}\right] + \left[\begin{array}{cc} v_j & w_j \end{array}\right] T_i\right) \left[\begin{array}{c} t_{h-1}^{\sigma} \\ g_{h-1}^{\sigma} \end{array}\right]$
4. **Select** the child classifier with the highest U_i as **trusted child**.
5. Compute the corresponding $\left[\begin{array}{cc} v_i & w_i \end{array}\right]$ and **transmit** it to a previous classifier who requested it.

4.3 Decentralized ordering and operating point selection algorithm

In case of unfixed operating points, the local utility of classifier $C_i = C_{\sigma(h)}$ also depends on its local operating point x_i – but it does not directly depend on the operating points of other classifiers.

As a consequence, we can easily adapt the Partial Search Ordering Algorithm into a Partial Search Ordering and Operating Point Selection Algorithm by computing the maximal utility (in terms of x_i) for each child:

$$U_i(C_j) = \max_{x_i} \left(-\left[\begin{array}{cc} \rho_{\sigma(i)} & 0 \end{array}\right] + \left[\begin{array}{cc} v_j & w_j \end{array}\right] T_i(x_i)\right) \left[\begin{array}{c} t_j \\ g_j \end{array}\right] \quad (8)$$

To solve the local optimization problem (8), each classifier can either derive the nullity of the gradient if the DET curve function $f_i : p^F \mapsto p^D$ is known, or search for optimal operating point using a dichotomy method (since $U_i(C_j)$ is concave). Such methods have been well-detailed in [8] and the reader is referred to this reference for further details.

4.4 Robustness of the Partial Search Ordering Algorithm

Convergence of Partial Search Algorithm.
– Under stable conditions the Partial Search Algorithm converges to the equilibrium point of the stochastic decision process.
– For fixed operating point the Partial Search Algorithm converges to the optimal order if $p_j^i > 0 \; \forall \, i, j$.

Proof. We consider a stable input stream, i.e. with non varying characteristics. Each iteration of the Global Partial Search Iteration will be indexed by the superscript by $(\mathfrak{t})$. Convergence of Partial Search algorithm can be obtained by proving recursively that each iteration improves the global utility of the stream mining system. Indeed, since $(x_i^{(\mathfrak{t}+1)}, C_{i\rightarrow}^{(\mathfrak{t}+1)}) = \underset{C_j \in Children(C_i)^{(\mathfrak{t}+1)}}{\arg\max} U_i(C_j)$, we can derive recursively that $\forall h : U_{\sigma(h)}^{(\mathfrak{t}+1)} \geq U_{\sigma(h)}^{(\mathfrak{t})}$, and in particular, this is true for the source classifier, which corresponds by construction to the system's utility. Increasing and upper-bounded, the global utility thus converges. The limit is thus obtained at an equilibrium point of the decision process, since by construction, for these stream characteristics, each classifier cannot find a better operating point nor forward the stream to a child classifier, without having a loss of utility.

We also want to prove that in case of fixed Operating Point, the algorithm will always converge to the optimal order. Let $P(N) = \mathbb{P}$(For any N classifiers, the Partial Search Ordering Algorithm converges to optimal order). Denote by $\sigma^{(\mathfrak{t})}$ the order provided by the algorithm at time $\mathfrak{t}$ and σ_{opt} the optimal order of this chain of classifiers. Then $P(N) = \mathbb{P}(\exists \mathfrak{t} \text{ s.t. } \sigma^{(\mathfrak{t})}(1) = \sigma_{opt}(1))P(N-1)$, where, $\mathbb{P}(\exists \mathfrak{t} \text{ s.t. } \sigma^{(\mathfrak{t})}(1) = \sigma_{opt}(1)) = \lim_{\mathfrak{t}\rightarrow\infty} \sum_{t=1}^{\mathfrak{t}} p^0_{\sigma_{opt}(1)}(1 - p^0_{\sigma_{opt}(1)})^t$. Since $p^0_{\sigma_{opt}(1)} > 0$, the geometric sum converges and $P(N) = 1 * P(N-1) = \ldots = P(1) = 1$, which ends the proof $\square$

In case of joint ordering and operating point selection, there exist multiple equilibrium points, each corresponding to a local minimum of the utility function. The selection of the equilibrium point among the set of possible equilibria depends on the initial condition (i.e. order and operating points) of the algorithm. We can perform the Partial Search Algorithm for multiple initial conditions and select the equilibrium yielding the maximum utility.

Convergence speed. In practice, stable stream condition will not be verified by the stream mining system, since the system's characteristics vary at a time scale of τ^{dyn}. Hence, rather than achieving convergence, we would like the Partial Search Algorithm to reach near-equilibrium fast enough for the system to deliver solution to the accuracy and delay joint optimization on a timely basis.

5. PARAMETRIC PARTIAL SEARCH ORDER AND OPERATING POINT SELECTION ALGORITHM

In this section, we aim to construct an algorithm which would maximize as fast as possible the global utility of the stream mining system expressed in (4). We want to determine whether it is worth for a classifier C_i to probe a child classifier C_j for its utility parameters and determine search probabilities p_j^i of the Partial Search Algorithm accordingly.

5.1 Tradeoff between efficiency and computational time

Define an experiment $E_{i\to j}$ as classifier C_i's action of probing a child classifier C_j by requesting its utility parameter $\left[\begin{array}{cc} v_j & w_j \end{array}\right]$. Performing an experiment can lead to a higher utility, but will induce a cost in terms of computational time:

- Let $\hat{U}(E_{i\to j}|s_k)$ be the expected additional utility achieved by the stream mining system if the experiment $E_{i\to j}$ is performed under state s_k.
- Let τ^{ex} represent the expected amount of time required to perform an experiment.

Then, the total expected utility per iteration is given by $\hat{U}(p_j^i) = \sum p_j^i \hat{U}(E_{i\to j}|s_k)$ and the time required for one iteration is $\tau(p_i^j) = \hat{n}(p_j^i)\tau^{ex}$, where $\hat{n}(p_j^i)$ represents the expected number of experiments performed in one iteration of the Partial Search Algorithm.

The allocation of the screening probabilities p_j^i aims to maximize the total expected utility within a certain time:

$$\left\{\begin{array}{ll} \underset{p_j^i\in[0,1]}{\text{maximize}} & \hat{U}(p_j^i) \\ \text{subject to} & \tau(p_i^j) \leq \mathfrak{C}\tau^{dyn} \end{array}\right. \tag{9}$$

5.2 How much to search? – Efficiency VS Flexibility

Suppose for the moment that we perform random search: any classifier C_i searches through its children with the same probability $p_j^i = p$:

- $p \approx 0$ corresponds to algorithms which seldom perform new search and thus screen only a few branches of the classifiers' tree at each iteration. Hence, reaching the optimal order requires on average $1/p^N \to \infty$ iterations, each with complexity $O(N!p^N \to 0)$.
- $p \approx 1$ corresponds to algorithms which screens through nearly all the branches of the classifier tree at each iteration. Hence, reaching the optimal order requires on average $1/p^N \approx 1$ slow iterations, each with complexity $O(N!p^N \approx O(N!))$.

Performance of search algorithm for random search. For a given p, the average number of children that classifier $C_{\sigma(h)}$ probes is then

$$n_h = \sum_{k=0}^{N-k} k \underbrace{\left(\begin{array}{c} k \\ N-h \end{array}\right) p^k (1-p)^{N-h-k}}_{\mathbb{P}(k \ children \ probed)} = (N-h)p$$

The average number of orders explored per iteration will thus be $N_O = \prod_{h=0}^{N-1} n_h = N!p^N$ and the processing time required will be $\tau(p) = \tau^{ex} \sum_{h=0}^{N-1} \left(\prod_{i=0}^{h} n_i\right) = \tau^{ex} \sum_{h=0}^{N-1} \frac{N!}{(N-h)!} p^h$.

Speed of search as a modeling variable for the expected utility. Since the utility $\hat{U}$ expected from a set of experiments $(E_{i\to j})$ cannot be determined analytically, we can monitor performance by modeling $\hat{U}$ by the speed of search V, defined as the average number of orders explored in a certain amount of time.

$$V(p) = \frac{N_O}{\tau} = \left(\tau^{ex} \sum_{h=1}^{N} \frac{1}{(N-h)!p^{N-h}}\right)^{-1}$$

This modeling choice corresponds to the assumption that expected utility is proportional to the speed of search. Doing so, the optimization problem introduced in Eq. (9) becomes

$$\left\{\begin{array}{ll} \underset{p\in[0,1]}{\text{maximize}} & V(p) \\ \text{subject to} & \tau(p) \leq \mathfrak{C}\tau^{dyn} \end{array}\right. \qquad \mathfrak{C} << 1 \tag{10}$$

Since both the objective function and the constraint function are increasing in p, the optimal average screening p^* will be obtained by saturating the time constraint: $\tau(p) = \mathfrak{C}\tau^{dyn}$. Hence the Partial Search Algorithm with constant screening probability $p_j^i = p^* = \tau^{-1}(\mathfrak{C}\tau^{dyn})$ enables to scout at each iteration as many orders as possible within a time window of average size $\mathfrak{C}\tau^{dyn}$.

5.3 Where to search? – Exploration VS Exploitation

We refine the speed-driven optimization for the selection of the average probability by additionally keeping record of the local utilities achieved in the past by classifier and taking decisions accordingly. Given an average search probability p, we can encourage *exploring* unprobed children or *exploit* already-visited orders, by weighting the propensity of classifier C_i to probe one of its children classifier C_j, based on past experiments' reward.

Learning algorithm. We define the probing probability as

$$p_j^i = p^{\sigma(h)} \frac{e^{\beta U_i(j,k)}}{\sum_{C_l \in Children(C_i)} e^{\beta U_i(l,k)}} \tag{11}$$

where the utility $U_i(j,k)$ represents the latest local utility achieved by classifier C_i when it transmitted the stream to classifier C_j, given that $\theta_i \in S_k$[2], and is updated on-the-go.

[2]The number of bins L used to described the continuous segment $[1,\infty]$ as a finite set of points and the quantization chose have important impact on the performance of the learning algorithm: higher number of states enable a more tailored decision, but since states will be more seldom visited, past utilities will be updated less often

In practice, the weight associated to a specific child based on its past reward could be determined using any increasing function f. Using $f_\beta(U) = \frac{e^{\beta U}}{\sum_V e^{\beta V}}$ is motivated by the analogy of a classifiers utility U to an energy [9]. In this case, $\frac{e^{\beta U}}{\sum_V e^{\beta V}}$ represents the equilibrium probability of being at an energy level U. As such, the parameter β can be interpreted as the inverse of a temperature, i.e it governs the amount of excitation of the system:

- $\beta = 0$ corresponds to a very excited system with highly time-varying characteristics. In this case, since characteristics change very quickly, random exploration: $p_j^i = p$ is recommended by the algorithm.
- $\beta = \infty$ corresponds to a non-varying system. Then, full-exploitation of past rewards is recommended (given that all states were explored at least once) and weight should be concentrated only on the child which provides the maximum utility.
- $0 < \beta < \infty$ is a tradeoff between exploration ($\beta = 0$) and exploitation ($\beta = \infty$) and corresponds to settings where algorithmic search and environment evolution are performed at the same time scale ($\tau \approx \mathfrak{C}\tau^{dyn}$).

6. EXPERIMENTAL RESULTS

6.1 Accuracy VS Delay tradeoff analysis

Upper bound of delay cost λ_{max}. Since the system's global utility is bounded, we can show that for $\lambda > \lambda_{max} = \frac{c^M}{\sum_{k=1}^N \alpha_k - \min(\alpha_k)/\Phi}$, the stream should not be processed: the first classifier would then be the one with minimal processing cost and would drop all tuples.

Impact of cost parameters. We consider $N = 4$ classifiers $C_1, ..., C_4$ with different processing costs α and different DET curves, characterized by their Area Under Graph (AUG). We quantize the DET curve and keep 4 operating points $x_i = 0, 1/3, 2/3, 1$. For this experiment, we choose classifiers with increasing α and decreasing AUG, as shown in Data Table 1, and fix equal *a-priori* selectivities ϕ and $\overline{\phi}$. With this choice of parameters, we expect classifiers to be ordered increasingly for high relative delay penalty or decreasingly for high accuracy weight.

	α	AUG
C_1	.25	.82
C_2	.5	.87
C_3	.75	.93
C_4	1	.98

Data Table 1

The results in Table 1 are obtained by computing the utility achieved for every order and operating point and keeping the optimal setup. We verify that classifiers are ordered according to their accuracy (AUG) for $\lambda = 0$ and according to their processing time α when delay is taken into account. Furthemore, we can notice that as the misclassification cost c^F becomes more important, classifiers choose operating points with low probability of false alarm.

$c^M = 10$		$c^F = 0$	$c^F = 1$	$c^F = 3$
$\lambda = 0$	σ	$[C_4\ C_3\ C_2\ C_1]$	$[C_4\ C_3\ C_2\ C_1]$	$[C_4\ C_3\ C_2\ C_1]$
	$\mathbf{x}$	$[x_4\ x_4\ x_4\ x_4]$	$[x_4\ x_4\ x_2\ x_2]$	$[x_3\ x_2\ x_2\ x_2]$
$\frac{\lambda_{max}}{30}$	σ	$[C_1\ C_4\ C_3\ C_2]$	$[C_1\ C_2\ C_3\ C_4]$	$[C_1\ C_2\ C_3\ C_4]$
	$\mathbf{x}$	$[x_4\ x_4\ x_4\ x_4]$	$[x_4\ x_3\ x_2\ x_2]$	$[x_3\ x_2\ x_2\ x_2]$
$\frac{\lambda_{max}}{10}$	σ	$[C_1\ C_4\ C_3\ C_2]$	(C_1,x_1)	(C_1,x_1)
	$\mathbf{x}$	$[x_2\ x_2\ x_3\ x_4]$	No trans.	No trans.

Table 1: Optimal configuration for varying c^M and λ

and stored values might be obsolete for time-varying settings. Taking advantage of this tradeoff can be done through methods such as prioritized sweeping [10], to which we refer the interested reader.

6.2 System compliance of joint algorithms

For joint ordering and configuration, the optimal order cannot be achieved in a timely fashion. Indeed, for N classifiers with A operating points each, $O(N!A^N)$ iterations are required to determine the optimal order and operating points. Even though the joint algorithms exposed can only be proved to converge to local optima (due to non convexity of utility function), they enable to obtain a suboptimal order and operating points within a fixed amount of time.

To cope with the stream mining environment, algorithms must update their results on regular time basis: we are thus interested in the amount of time required by one iteration of the Partial Search algorithm, shown on Table 2 for 20 classifiers, each having 100 operating points[3].

	$p_j^i = .01$	$p_j^i = .03$	$p_j^i = .05$
τ	0.34s	5.3s	37s

Table 2: Computational time achieved for various p

The Partial Search algorithm makes sure that each iteration is performed in a sufficiently small amount of time to ensure that it arrives on time: $\tau \leq \mathfrak{C}\tau^{dyn}$ by adjusting the parameter p.

6.3 Decentralized convergence rate

Figure 5 compares the utility achieved by the Partial Search algorithm for varying values of p, with $N = 20$ and $A = 100$ actions.

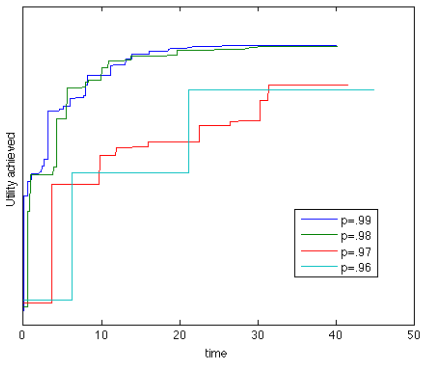

Figure 5: Joint algorithms comparison

By construction, the Partial Search Algorithm is non increasing. The parameter p enables to control the time per iteration. As such, small values of p correspond to small stair steps, while higher value of p correspond to stairs with only a few large steps.

7. EXTENSION: PARALLEL TOPOLOGIES FOR MULTI-NODE CLASSIFIER

The proposed ordering algorithm apply to a chain of classifiers ordered linearly. Yet, classifier chains are not necessarily optimal for classifiers distributed on multiple resource-constrained processing nodes. Indeed, simultaneously processing the stream in parallel on multiple classifiers chains could reduced the total delay, thus improving global performance. In this section, we provide a extension framework for alternate non-linear topologies and discuss the number of chains to be used under simplifying assumptions.

[3]Classifier's characteristics (p^F, p^D), ϕ and α were generated randomly. $c^M = 10$, $c^F = 1$, $\lambda = 0.1$

7.1 Resource constrained stream mining

Resource constraints. A key challenge in distributed real-time stream mining systems arises from the need to cope effectively with system overload, due to large data volumes and limited system resources (e.g. CPU, memory, I/O bandwidth) [1]. Specifically, there is a large computational cost incurred by each classifier (proportional to the data rate) that limits the rate at which the application can handle input data.

Let N classifiers be instantiated on M processing nodes[4], each of which has a given available resource r_m^{max}.

Message exchange. In previous sections, we did not take into consideration the time required by classifiers to communicate with each other. If classifiers are all grouped on a single computer, such communication time can be neglected compared to the time required by classifiers to extract data features. However for classifiers instantiated on separate nodes, this communication time can greatly increase the total time required to deal with a stream tuple.

As such, we would like to limit the communication between nodes, ie:

- Limit sending the data and control messages across nodes.
- Limit the message exchanges required by the decentralized algorithm ($\begin{bmatrix} v & w \end{bmatrix}$) and by the load-shedding ratios γ learning.

Parallel processing of a stream. When classifiers are placed across multiple nodes, one way to minimize transmission (of the stream and messages) across processing nodes we can organize classifiers within each node into local chains, and connect only the root and leaf of this chain to classifiers on other nodes. This is equivalent to imposing local constraints on the order selection process. In general, for M nodes, this will lead to M such chains, with the data stream being forwarded across from one node to another only M-1 times.

Alternately, we can also consider a parallel processing scenario, where all classifiers within one node are organized into a chain, and data is processed in parallel through the different chains. Since we are interested only in a conjunction of the output of all classifiers, the results of the multiple chains can be fused (i.e. select only data passed by all chains) to get the final result. Note that this parallel processing requires replicating the stream across the nodes, thereby leading to increased resource consumption, however it also leads to reduced end-to-end latency.

Examples of the serial and parallel processing topologies are shown in Figure 6.

In practice, the optimal topology selection may be a combination of the serial and parallel topologies based on the communication time between nodes, their processing capabilities etc.

In the following paragraph, we are going to model the stream mining system in order to encompass such non-linear topology. We highlight the intuitions justifying the use of parallel chain topologies and propose a method for determining such topology under simplifying assumptions. This is included as a motivation for future work.

7.2 Global utility of a set of classifiers for multi-node stream mining system

[4]The placement of classifiers across nodes is outside the scope of this paper.

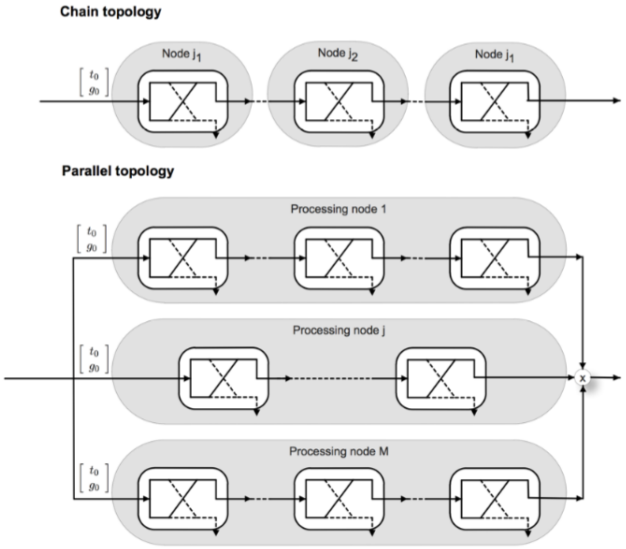

Figure 6: Stream processing through multiple chains

Single chain We model the average time needed for classifier C_i to extract the feature of a stream tuple as α_i^{feat} (in seconds) and the average time needed to send a stream tuple from classifier C_i to classifier C_j as $\alpha_{i,j}^{com}$ (in seconds). The matrix α^{com} can be supposed as symmetric. The average end-to-end processing time required by the σ-ordered chain to process stream data is

$$c_{delay}^{\sigma} = \sum_{k=1}^{N} \alpha_{\sigma(k)}^{feat} t_{k-1}^{\sigma} + \sum_{k=1}^{N-1} \alpha_{\sigma(k),\sigma(k+1)}^{com} t_k^{\sigma} \qquad (12)$$

Multiple chain Suppose that instead of considering classifiers in a chain, we process the stream through R chains, where chain r has N_r classifiers with the order $\sigma_r \in \mathcal{P}erm(N_r)$. The answer of the query is then obtained by intersecting the output of each chain and we neglect the time required for this operation.

The end-to-end processing time is then given by

$$c_{delay}^{\sigma} = \max_{1 \le r \le R} \left(\sum_{k=1}^{N_r} \alpha_{\sigma_r(k)}^{feat} t_{k-1}^{\sigma_r} + \sum_{k=1}^{N_r-1} \alpha_{\sigma_r(k),\sigma_r(k+1)}^{com} t_k^{\sigma_r} \right) \qquad (13)$$

The output of parallel processing chain is recomposed by intersection. As a consequence, we will make the approximation that the accuracy is independent of the number of chains carried out (the end-goodput is not modified, and, assuming independance of classifiers, niether would the end-throughput).

7.3 How to group classifiers within parallel chains of classifiers

Let's suppose that all classifiers have extraction time α^{feat}. Furthermore, we will consider that the communication time is either α_{int}^{com} for classifiers of a same node or α_{ext}^{com} for classifiers of different nodes.

Let's define the symmetric binary matrix $P = 1 - MM^T$.

$$P_{i_1,i_2} = 1 - \sum_{j=1}^{M} M_{i_1,j} M_{i_2,j} = \begin{cases} 0 & \text{if } C_{i_1}, C_{i_2} \text{ are on the same node} \\ 1 & \text{if } C_{i_1}, C_{i_2} \text{ are on different nodes} \end{cases}$$

$$\begin{aligned} \text{With } \alpha_{i_1,i_2}^{com} &= \alpha_{ext}^{com} P_{i_1,i_2} + \alpha_{int}^{com}(1 - P_{i_1,i_2}) \\ &= \alpha_{int}^{com} + P_{i_1,i_2}(\alpha_{ext}^{com} - \alpha_{int}^{com}) \end{aligned}$$

the end-to-end processing time can be rewritten as:

$$c^{\sigma}_{delay} = \max_{1\leq r\leq R}\left[\alpha^{feat}\sum_{k=1}^{N_r} t^{\sigma_r}_{k-1} + \alpha^{com}_{int}\sum_{k=1}^{N_r-1} t^{\sigma_r}_{k} + (\alpha^{com}_{ext}-\alpha^{com}_{int})\left(\sum_{k=1}^{N_r-1} P_{\sigma(k),\sigma(k+1)} t^{\sigma_r}_{k}\right)\right]$$

By indexing the chain responsible of maximal processing time as r_0, the delay can be expressed as:

$$c^{\sigma}_{delay} = \alpha^{feat}t_0 + (\alpha^{feat}+\alpha^{com}_{int})\sum_{k=1}^{N_{r_0}-1} t^{\sigma_{r_0}}_{k} + (\alpha^{com}_{ext}-\alpha^{com}_{int})\sum_{k=1}^{N_{r_0}-1} P_{\sigma_{r_0}(k),\sigma_{r_0}(k+1)} t^{\sigma_{r_0}}_{k}$$

Instinctively, we want to group classifiers instantiated on a same node on a same chain in order to limit the stream being sent back and forth from one node to the other. Indeed, with chains grouping classifiers instantiated on a same node, $P_{\sigma_r(k),\sigma_r(k+1)} = 0$ and the total delay is minimized.

7.4 Determination of number of chains

Expected delay. Suppose that N classifiers on M nodes process data in parallel through R streams. Since we want to limit the maximum delay, we are going to choose chains with more or less the same number of classifiers. We will therefore model that each chain groups N/R classifiers.

Furthermore, since we group – as much as possible – classifiers instantiated on a same node on a common chain, the average number of nodes per chain is M/R. Stream will be send from one node to another $M/R-1$ times in average: the probability of consecutive classifiers of a chain belonging to the different nodes is then $\mathbb{P}(P_{i,j}{=}1) = \frac{M/R-1}{N/R-1} = \frac{M-R}{N-R}$. Hence, with $\Psi = \sum_{k=1}^{N_r-1} t^{\sigma_r}_k$,

$$\mathbb{E}(c^{\sigma}_{delay}) = (t_0+\Psi)\alpha^{feat} + (\Psi\frac{M-R}{N-R})\alpha^{com}_{ext} + \underbrace{(\Psi\frac{N-M}{N-R})\alpha^{com}_{int}}_{\approx 0}$$

Observe that in practice, $\alpha^{com}_{int} << \alpha^{com}_{ext}, \alpha^{feat}$ and we will suppose it to be zero.

Expected resource constraints. The total amount of resource used by the stream mining system is given by

$$r_{tot} = \sum_{j=1}^{M} r_j = \sum_{r=1}^{R}\alpha^{feat}\sum_{k=1}^{N_r} t^{\sigma_r}_{k-1} \approx R\alpha^{feat}\sum_{k=1}^{N/R} t^{\sigma_r}_{k-1}$$

It is clearly increasing with the number of chains R.

Tradeoff between extraction time and communication time. We can thus observe a tradeoff between extraction time α^{feat} and communication time α^{com}:

- If $\alpha^{com} << \alpha^{feat}$, there should be only one chain, so that lower classifier only process a small fraction of the whole data and in order to reduce the resource constraints.
- If $\alpha^{feat} << \alpha^{com}$, then there should be as many chains as possible, i.e one chain per node. Indeed, in this case, the stream would never be sent from one node to the other. Observe that the order of classifiers within each node is determined using Partial Search Algorithm.
- $\alpha^{feat} \sim \alpha^{com}$ represents a tradeoff situation where there should be $1 \leq R \leq M$ chains, each chain grouping classifiers belonging to a common node.

8. CONCLUSION

In this paper, we design distributed and adaptive algorithms to determine the optimal linear topology of a set of classifiers to trade off filtering accuracy and processing delay. These algorithms are designed for large scale multimedia analysis applications that require the detection of multiple concepts from high volume multimedia data using a set of distributed processing resources. We also extend our algorithms to optimize the configuration of individual classifiers in the selected topology, by determining their optimal operating points under the given data characteristics, classification performance, and delay constraints. In our decentralized approach individual classifiers make autonomous decisions on what operating point (e.g. filtering threshold) to use while classifying data, as well as which classifier to forward the data to, after processing, i.e. distributedly determining a topology order. We use a Parametric Partial Search Algorithm, that dynamically searches through a subset of all orders in each iteration, to tradeoff the optimality of the solution with the time required for convergence. Our experimental results shows how the screening parameter p enables to realize such tradeoff. Finally, the proposed algorithms are extended to consider resource-constraints, where a multi-chain stream mining system can be preferred.

9. REFERENCES

[1] B. Babcock, S. Babu, R. Motwani, and M. Datar. Chain: operator scheduling for memory minimization in data stream systems. In *ACM SIGMOD*, 2003.

[2] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelined stream filters. *ACM SIGMOD international conference on Management of data*, 2004.

[3] R. Ducasse, D. S. Turaga, and M. van der Schaar. Adaptive topologic optimization for large-scale stream mining. *IEEE Journal on Selected Topics in Signal Processing*, under review.

[4] B. Foo and M. van der Schaar. A distributed approach for optimizing cascaded classifier topologies in real-time stream mining systems.

[5] F. Fu, D. S. Turaga, O. Verscheure, M. van der Schaar, and L. Amini. Configuring competing classifier chains in distributed stream mining systems. *IEEE Journal on Selected Topics in Signal Processing*, 2007.

[6] R. Lienhart, L. Liang, and A. Kuranov. A detector tree of boosted classifiers for real-time objects detection and tracking. In *Proceeedings of the International Conference on Multimedia and Expo (ICME)*, 2003.

[7] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting spam web pages through content analysis. In *Proceedings of the 15th international conference on World Wide Web*, May 2006.

[8] H. Park, D. S. Turaga, O. Verscheure, and M. van der Schaar. Foresighted tree configuring games in resource constrained distributed stream mining systems. *IEEE Int. Conf. on Acoustics, Speech, and Signal Process*, 2009.

[9] L. Saul and M. I. Jordan. Learning in boltzman trees. *Neural Computation*, 1994.

[10] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. *The MIT Press, Cambridge, MA,*, 1998.

[11] V. Vazirani. *Approximation algorithms*. Springer Verlag.