Adaptive Topologic Optimization for Large-Scale Stream Mining

Raphael Ducasse, Deepak S. Turaga, and Mihaela van der Schaar

***Abstract*—Real-time classification and identification of specific features in high-volume data streams are critical for a plethora 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 focus on selecting the optimal topology (chain) of classifiers, and present algorithms for classifier ordering and configuration, to tradeoff accuracy of feature identification with filtering delay. The order selection is dependent on the data characteristics, system resource constraints as well as the performance and complexity characteristics of each classifier. We first develop centralized algorithms for joint ordering and individual classifier operating point selection. We then propose a decentralized approach and use reinforcement learning methods to design a dynamic routing based order selection strategy. We investigate different learning strategies that lead to rapid convergence, while requiring minimum coordination and message exchange.**

***Index Terms*—Classifier topology construction, large-scale stream mining, multi-concept detection, optimization.**

I. Introduction

THE spread of computing, authoring and capturing devices along with high-bandwidth connectivity has led to a proliferation of streaming data including documents, e-mail, transactions, digital audio, video and images, sensor measurements etc. As a consequence, there is a large class of emerging applications for annotation, online search and retrieval, and knowledge extraction that require the analysis of these high-volume data streams in real-time. Examples of these applications include online financial analysis, fraud detection, photo and video streaming, spam classification, medical services, search engines [15], [17], [20], [24], etc. Each application can be viewed as a processing pipeline that analyzes streaming data from a set of raw data sources to extract valuable information in real-time.

The development of large-scale stream mining platforms [1], [13] has enabled applications to be constructed as topologies of distributed operators deployed on a set of heterogeneous processing resources. This has led to enhanced scalability, reliability, and several performance-complexity tradeoffs—that allow the achievement of desired performance goals. Specifically, stream mining applications are constructed using a topology of low-complexity binary classifiers, each performing feature extraction and classification specific to different concepts of interest. This allows successive identification of multiple features in the data, and provides significant advantages in terms of reduced resource consumption through appropriate dynamic data filtering.

In this paper, we focus on constructing and ordering such classifier topologies tailored towards answering specific queries of interest on data streams. These queries are posed as a set of conjunctive filtering/classification operations, as is common in search and knowledge extraction applications (e.g., find data that has attribute A and attribute B but not attribute C) [14]. The focus of our work is on designing topologies that minimize the end to end misclassification of data, while also minimizing the end to end processing delay, in resource-constrained stream mining settings.

This work lies at the intersection of research in large-scale stream mining system configuration and analysis, and of the linear topology ordering problems. Related work on stream mining optimization [2], [6], [25], [26] is focused on resource allocation and load-shedding approaches. There has also been some work on optimizing fixed topologies of classifiers under resource constraints [9], [21]. However, this related research ignores the classifier ordering problem: the topology is assumed fixed and determined *a priori*.

There has been some related work on ordering conjunctive filtering operations on streaming data, as part of the broader pipeline-ordering problem [5], [7], [11], [22] as well as the classical set-cover problem [16], [30]. These approaches design low-complexity algorithms to order a set of filters applied to streaming data, to minimize the end-to-end processing time. They use centralized greedy optimization techniques to derive the appropriate order, and develop bounds on performance, in terms of approximation factors to the utility of the optimal solution. However, they always consider perfect classifiers, i.e., classifiers that make no mistakes, and neglect computational and network resource constraints. Additionally, their design does not support dynamic ordering for adaptation to frequent and abrupt environment characteristics—inherent to multi-query scenarios with asynchronous classifiers.

Instead, in this paper, we consider the problem of topology construction for imperfect classifiers deployed on a resource-constrained stream mining system. The key contributions of

Manuscript received April 21, 2009; revised November 05, 2009; accepted November 11, 2009. First published April 15, 2010; current version published May 14, 2010. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Are Hjørungnes.

R. Ducasse and M. van der Schaar are with the Department of Electrical Engineering, University of California at Los Angeles (UCLA), Los Angeles, CA 90095-1594 USA (e-mail: ducasse@ucla.edu; raphael.ducasse@gmail.com; mihaela@ee.ucla.edu).

D. S. Turaga is with IBM T. J. Watson Research Center, Hawthorne, NY 10532 USA (e-mail: turaga@us.ibm.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSTSP.2009.2039180

this paper areas follows. We formalize the topology construction problem for query optimization in stream mining systems with imperfect classifiers. We formulate this as an optimization problem with a joint objective to maximize the classification accuracy and minimize processing delay. We develop a centralized Greedy Algorithm for order selection—with bounds on performance—and a combined sequential quadratic programming (SQP)-Greedy Algorithm for joint order selection and classifier configuration. We then develop decentralized topology construction algorithms, where classifiers make dynamic routing decisions independently and asynchronously based on local information and objectives. We use reinforcement learning-based methods to examine tradeoffs between algorithm convergence, and solution optimality.

This paper is organized as follows. In Section II, we provide a background on classifiers used for concept detection and underline the accuracy tradeoff between detection and false alarm. We then extend these notions to networked topologies of classifiers. In Section III, we formulate the order and operating point selection optimization problem. In Section IV, we propose a centralized solution and derive bounds on performance. In Section V, we discuss the limitations of a centralized approach and present our distributed optimization based approaches in Section VI and Section VII. We present experimental results in Section VIII and conclusions in Section IX. All proofs are included in the Appendix A.

II. Binary Classification

A. Modeling Binary Filters

A binary classifier C labels input data into two classes $\mathcal{H}$ (class of interest) and $\overline{\mathcal{H}}$. 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* selectivity $\phi = \mathbb{P}(X \in \mathcal{H})$, i.e., the *a priori* probability that the data belongs to the class of interest. Correspondingly, $1 - \phi = \mathbb{P}(X \in \overline{\mathcal{H}})$. The *a priori* selectivities ϕ are computed on a training and cross-validation data set. For well-trained classifiers, it is reasonable to expect that the performance on new, unseen test data is similar to that characterized on training data. In practice, there is potential train–test mismatch in behavior, but this can be accounted for using periodic reevaluation of the classifier performance (using potentially feedback on generated results) [19].

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 $p^D = f(p^F)$ that is increasing between, 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. In practice, each classifier has a fixed number of possible operating points obtained by quantizing the DET curve.

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). For unit input rate, these may be derived as

$$\begin{aligned} t &= \phi p^D + (1-\phi)p^F \\ g &= \phi p^D. \end{aligned} \tag{1}$$

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 (text classification) to multiple seconds (for complex image classification).

B. Classifier Chain

Stream data analysis applications pose queries on data that require multiple concepts to be identified. For instance, to identify images containing scenes from people on a beach, it may be necessary to identify the semantic concepts "ocean," "sand," "people," etc., with separate classifiers, and data of interest needs to satisfy a conjunction of these classifiers. By partitioning the problem into a set of classifiers and filtering data (i.e., discarding data that is labeled as not belonging to a class of interest) successively; we can control the amount of resources consumed by each classifier in the ensemble. Hence, the the "sand" classifier may be used to filter data before the "ocean" classifier. This results in lower complexity processing, because the "ocean" classifier needs only to process a subset of images. In this paper, we focus on chain/linear topologies of classifiers as these form the basic building blocks for several other topologies (e.g., trees) and map directly onto conjunctive filtering operations [14], [16]. We intend to extend this work to consider arbitrary topologies, including trees and directed acyclic graphs as part of future work. We now define the throughput and goodput for a chain of classifiers.

Queries and Classifier Chain: Consider a query q that is answered as a conjunction of a set of N classifiers $\mathcal{C}(q) = \{C_1, \ldots, C_N\}$. As an example, these classifiers may be arranged into a chain where the output of classifier C_{i-1} feeds the input of classifier C_i and so on.

Conditional a priori Selectivity: We define the *a priori* positive selectivity as the conditional probability of data belonging to classifier C_i's class of interest, given that it belongs to the class of interest of the previous $i-1$ classifiers

$$\phi_i = \mathbb{P}\left(X \in \mathcal{H}_i | X \in \bigcap_{k=1}^{i-1} \mathcal{H}_k\right).$$

Similarly, we define the negative *a priori* selectivity as the conditional *a priori* probability of data belonging to the class of interest for classifier i given that it does not belong to the class of interest for at least one of the $i-1$ preceding classifiers: $\overline{\phi_i} = \mathbb{P}(X \in \mathcal{H}_i | X \notin \bigcap_{k \le i-1} \mathcal{H}_k)$. 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.

Throughput and Goodput: Let t_i and g_i denote the throughput and goodput of classifier C_i. As in [27], [28],

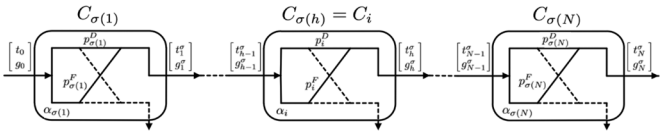


Fig. 1. Classifier chain.

the throughput and goodput of the stream is computed by each classifier recursively as follows:

$$\begin{bmatrix} t_i \\ g_i \end{bmatrix} = \underbrace{\begin{bmatrix} a_i & b_i \\ 0 & c_i \end{bmatrix}}_{T_i^{i-1}} \begin{bmatrix} t_{i-1} \\ g_{i-1} \end{bmatrix} \tag{2}$$

where $a_i = p_i^F + (p_i^D - p_i^F)\overline{\phi_i}$, $b_i = (p_i^D - p_i^F)(\phi_i - \overline{\phi_i})$, and $c_i = p_i^D \phi_i$. Details of this derivation can be found in Appendix A, Result no 1.

Independent Classifiers: For a set of independent classifiers, the positive and negative *a priori* selectivities are equal: $\phi_i = \overline{\phi_i} = \mathbb{P}(X \in \mathcal{H})$. As a consequence, the transition matrix is diagonal, i.e., $b_i = 0$.

C. Ordered Classifier Chain

Let $\mathcal{P}erm(N)$ represent the set of bijections of $[1, N]$: $\mathcal{P}erm(N) = \{\sigma | \forall j \in [1, N], \exists i \text{ such that } \sigma(i) = j\}$.

Definition: Classifier Order: An order of a set of classifiers $\{C_1, \ldots C_N\}$ 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 hth classifier in the chain is C_i. To illustrate the different notations used, a σ-ordered classifier chain is shown in Fig. 1.

With the *a priori* conditional selectivities $\phi_h^\sigma = \phi_{\sigma(h)}$, and $\overline{\phi_h^\sigma} = \overline{\phi_{\sigma(h)}}$, as defined earlier, the throughput and goodput of classifier $C_{\sigma(h)}$ are given by $\begin{bmatrix} t_h^\sigma \\ g_h^\sigma \end{bmatrix} = T_h^\sigma \begin{bmatrix} t_{h-1}^\sigma \\ g_{h-1}^\sigma \end{bmatrix}$.

End-to-End Characteristics of an Ordered Chain of Classifiers: Recursively from classifier $C_{\sigma(1)}$ to $C_{\sigma(N)}$, the *a priori* probability of satisfying a query and the end-to-end throughput and goodput are given by

$$\Phi^\sigma = \prod_{h=1}^{N} \phi_h^\sigma \quad \text{and} \quad \begin{bmatrix} t_N^\sigma \\ g_N^\sigma \end{bmatrix} = T_N^\sigma \ldots T_1^\sigma \begin{bmatrix} t_0 \\ g_0 \end{bmatrix}. \tag{3}$$

III. Problem Formulation

In this section, we formulate a general optimization problem in terms of simultaneously minimizing a global misclassification penalty and a processing delay, by jointly selecting the order of classifiers within the chain as well as operating point for each individual classifier. We then analyze the properties of the defined utility function, and discuss the underlying subproblems of ordering and operating point selection.

A. Cost Definitions

Traditional *data mining* system minimize a misclassification cost defined as a combination of both a cost for missing query-answering tuples and a cost for wrongly accepting stream tuples which do not correspond to the query. However, since *stream mining* systems require delivering real-time results, we also need to monitor the delay between the moment data enters the stream mining system and the moment it comes out. The total cost is therefore a weighted sum of misclassification cost and delay cost.

- **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. Hence, the total misclassification cost is

$$c_{\text{err}}^\sigma = c^M \underbrace{(\Phi t_0^\sigma - g_N^\sigma)}_{\text{missed data}} + c^F \underbrace{(t_N^\sigma - g_N^\sigma)}_{\text{false alarms}}. \tag{4}$$

- **Processing delay cost**. Delay may be defined as the time required by the chain of classifiers to process a stream tuple. Let $\alpha_{\sigma(h)}$ denote the expected processing time of the hth classifier $C_{\sigma(h)}$. For an order σ, the average time required by the hth classifier to process a stream tuple is given by $\delta_h^\sigma = \alpha_{\sigma(k)} P_h^\sigma$, where P_h^σ denotes the fraction of data which has not been rejected by the first $h-1$ classifiers and still needs to be processed through the remaining classifiers of the chain. Recursively, $P_h^\sigma = \prod_{i=1}^{h-1} (t_i^\sigma / t_{i-1}^\sigma) = (t_{h-1}^\sigma / t_0)$. Hence, the expected end-to-end processing time for a σ-ordered chain is

$$c_{\text{delay}}^\sigma = t_0 \sum_{k=1}^{N} \delta_k^\sigma = t_0 \sum_{k=1}^{N} \alpha_{\sigma(k)} P_i^\sigma = \sum_{k=1}^{N} \alpha_{\sigma(k)} t_{k-1}^\sigma. \tag{5}$$

B. Optimization Problem Analysis

The utility function may be defined as the negative weighted sum of both the misclassification cost and the processing delay cost

$$U_\lambda = -c^M (\Phi^\sigma t_0 - g_N^\sigma) - c^F (t_N^\sigma - g_N^\sigma) - \lambda \sum_{k=1}^{N} \alpha_{\sigma(k)} t_{k-1}^\sigma. \tag{6}$$

The parameter λ controls the tradeoff between misclassification and delay and corresponds to the Lagrange multiplier of the dual objective maximization of U_λ) [3]. Physically, λ is the price of delay in misclassification units. This price parameter is adjusted by the system operator depending on its empirical requirements. Means to determine λ will be made explicit in Section VIII-A. The utility is a function of the throughputs and goodputs of the stream within the chain, and therefore implicitly depends on the following.

1) **The order σ in which the classifiers are arranged**: Different linear topologies lead to different delay penalties since the order impacts the total amount of data to be processed by the stream mining system. Furthermore, when classifiers are not independent, different orders lead to different misclassification penalties.
2) **The operating point $\mathbf{x}$ selected by each classifier**: We will characterize the choice of the operating point of classifier C_i on the DET curve by $x_i \in [0\,1]$, where $(p_i^F, p_i^D) = (x_i, f_i(x_i))$. We will refer to the vector of operating points as $\mathbf{x} = (x_1, \ldots, x_N) \in [0\,1]^N$.

Not all terms in the utility function U_λ depend on the order σ and the operating point $\mathbf{x}$ selected. In particular, ϕ^σ is a constant and, since transition matrices T_i are upper triangular, the goodput does not depend on the order of classifiers. Furthermore, when classifiers are independent, the transition matrices T_i are diagonal and therefore commute. As a consequence the end throughput $t_N(\mathbf{x})$ and goodput $g_N(\mathbf{x})$ are independent of the order. However, intermediate throughputs do depend on the ordering—leading to varying expected delays for the processing.

Canonic Optimization Problem: We may define the optimization of the utility in (6) as

$$\max_{\sigma,\mathbf{x}} U(\sigma,\mathbf{x}) = g_N(\mathbf{x}) - Kt_N^\sigma(\sigma,\mathbf{x}) - \sum_{k=1}^{N} \rho_{\sigma(k)} t_{k-1}^\sigma(\sigma,\mathbf{x}) \quad \text{subject to } 0 \le \mathbf{x} \le 1 \tag{7}$$

where $K = c^F/(c^F+c^M) \in [0,1]$ and $\rho_i = \lambda/(c^F+c^M)\alpha_i \in \mathbb{R}^{+N}$, respectively, represent the relative price of wrongly classifying a tuple and the relative price of classifier C_i delaying the answer to the query, relatively to missing data. Detailed descriptions of intermediate steps leading to this formulation are included in Appendix A, Result no 2.

Our optimization problem consists of two related subproblems.

- **Ordering**: Selecting the optimal order of the classifiers chain belongs to the set-cover problem class [22]. In general, there exist $N!$ different orders with *a priori* no relationship between their utilities. Hence, any search space increases combinatorially with N. This problem is exacerbated under dynamic settings where the optimal order has to be updated online, or in case of multiple queries, where each query has to be matched with a specific optimal order.
- **Operating point selection**: This problem has been well studied in [9] and [21], where solutions were designed using sequential quadratic programming to obtain local optima.

In the following sections, we first present a centralized approach to solve this twofold optimization problem and discuss the limits of such an approach. We then present decentralized algorithms that enable adaptation to dynamic environments through reinforcement learning techniques.

IV. Centralized Approach

In this section, we first solve the ordering subproblem through a Greedy Algorithm, with a performance that can be bounded analytically. We also extend the Greedy Algorithm to allow dynamic adaptation to time-varying characteristics of data and classifiers. We then present a joint optimization strategy which combines ordering with operating point selection strategies.

A. Centralized Ordering

For fixed operating points, the end goodput g_N is independent of the classifier order. As a consequence, we can simplify the formulation in (7) as

$$\underset{\sigma\in\mathcal{P}erm([1,N])}{\text{maximize}} \quad U_{\text{ord}} = -\left(\sum_{h=1}^{N} \rho_{\sigma(h)} t_{h-1}^\sigma + Kt_N^\sigma\right). \tag{8}$$

B. Greedy Algorithm

The Greedy Algorithm is based on the notion of *ex-post* selectivity. For a given order σ, we define the *ex-post* selectivity as the conditional probability of classifier $C_{\sigma(h)}$ labeling a data item as positive given that the previous $h-1$ classifiers labeled the data as positive,[1] i.e., $\psi_{\sigma(h)} = (t_h^\sigma/t_{h-1}^\sigma)$. The throughput at each step can be expressed recursively as a product of *ex-post* selectivities: $t_h^\sigma = \psi_{\sigma(h)} t_{h-1}^\sigma = \ldots = (\prod_{i=1}^{h} \psi_{\sigma(i)}) t_0$. The system utility can then be rewritten as $U_{\text{ord}} = -\sum_{i=0}^{N} \mu_i t_i^\sigma$, where μ_i^σ is defined as $\mu_i^\sigma = \begin{cases} \rho_{\sigma(i+1)} = \lambda(\alpha_{\sigma(i+1)}/(c^M+c^F)) & \text{if } i \le N-1 \\ K = c^F/(c^F+c^M) & \text{if } i = N \end{cases}$. The Greedy Algorithm then involves ordering classifiers in increasing order of ψ/μ. This intuitively corresponds to selecting classifiers with lower selectivity and smaller computational complexity earlier in the chain. We can then build an iterative solution as follows.

Centralized Algorithm 1 Greedy ordering

- Calculate the ratio $\psi_1^\sigma/\mu_1^\sigma$ for all N classifiers. Select $C_{\sigma(1)}$ as the classifier with lowest weighted non-conditional selectivity $\psi_1^\sigma/\mu_1^\sigma$. Determine $\begin{bmatrix} t_2^\sigma \\ g_2^\sigma \end{bmatrix}$.
- Calculate the ratio $\psi_2^\sigma/\mu_2^\sigma$ for all remaining $N-1$ classifiers. Select $C_{\sigma(2)}$ as the classifier with lowest weighted conditional selectivity $\psi_2^\sigma/\mu_2^\sigma$. Determine $\begin{bmatrix} t_1^\sigma \\ g_1^\sigma \end{bmatrix}$.
- Continue until all classifiers have been selected.

Theorem IV.1 (Performance of the Greedy Algorithm): The value of the utility U_{ord}^G obtained with the Greedy Algorithm's order is at least 1/4th of the value of the optimal order

$$\frac{1}{\kappa} U_{\text{ord}}^{\text{opt}} \le U_{\text{ord}}^G \le U_{\text{ord}}^{\text{opt}} \quad \text{with } \kappa = 4$$

The proof of this theorem is given in Appendix A, Result no 3. The approximation factor $\kappa = 4$ corresponds to a system with infinite number of classifiers [30]. In practice, this constant factor is smaller. Specifically, we have κ =2.35, 2.61, 2.8 for 20, 100, or 200 classifiers, respectively.

C. Adaptive Greedy Algorithm

Over time, conditional selectivities ϕ^σ and $\overline{\phi^\sigma}$ and input rate t_0 of the stream or classifiers processing complexities α may vary. Therefore, the order selection of classifiers may need to be performed dynamically. In this case, we derive an adaptive Greedy Algorithm which enables dynamic maintenance of the best classifier order without having to perform the complete reordering at each time step. The adaptive Greedy Algorithm, or A-Greedy is based on the following notion of greedy invariant.

Greedy Invariant: Let σ^G represent the order obtained by the Greedy Algorithm. $C_{\sigma^G(1)},\ldots,C_{\sigma^G(N)}$ satisfy the greedy invariant if and only if we have $\psi_{\sigma^G(j)} \le \psi_{\sigma^G(i)} \forall\, 1 \le i \le j \le N$. This is a condition on $O(N^2)$ of the $N!$ *ex-post* conditional

[1]Observe that for a perfect classifier ($p_{\sigma(h)}^D = 1$ and $p_{\sigma(h)}^F = 0$), the *a priori* conditional probability $\phi_{\sigma(h)}$ and the *ex-post* conditional probabilities $\psi_{\sigma(h)}$ are equal.

selectivities corresponding to the set of classifiers. The adaptive Greedy Algorithm then consists of the following two steps.

— Profiler: Update on-the-go the $O(N^2)$ conditional selectivities of the greedy invariant.

— Re-optimizer: If the greedy invariant is violated, switch the order of the two classifiers that violated it.

We refer the interested reader to [22] for further details on the adaptive Greedy Algorithm. However, notice that updating the *ex-post* selectivities requires strong coordination between classifiers. A first solution would be for classifiers to send their choice of operating point (p^F, p^D) to a central agent (which would also have knowledge about the *a priori* conditional selectivities ϕ^σ) and would compute the *ex-post* conditional selectivities. A second solution would be for each classifier C_i to send their rates t_i and g_i to the classifiers C_j which have not yet processed the stream for them to compute ψ_j^i. In both cases, heavy message exchange is required, which can lead to system inefficiency (cf. Section V-C). We will propose in Section VI a decentralized solution with limited message exchanges, as an alternative to the centralized approach.

Convergence of A-Greedy: In accordance with [22], we say that the stream and classifier characteristics have stabilized when the following three conditions hold over a long interval of time. 1)The data distribution of stream tuples is constant. 2) For every subset of classifiers $\{C_1, C_2, \ldots, C_n\} \subset \mathcal{C}$, the data distribution of input tuples passing all of the classifiers in the subset is constant. 3) The values of μ_i for each classifier C_i are constant. Under such stable conditions, it can be proved [22] that the adaptive Greedy Algorithm converges and has the same error bound as the Greedy Algorithm.

We can now build a joint algorithm to solve simultaneously the ordering and operating point selection problem.

D. Joint Order and Operating Point Selection

Operating Point Selection for a Fixed Order of Classifiers: Consider a fixed order of classifiers. Resource-constrained operating point selection for fixed topologies has been well studied [9], [21], [27]. The solutions proposed involve using iterative optimization techniques based on sequential quadratic programming (SQP) [3]. SQP is based on gradient-descent, and models a nonlinear optimization problem as an approximate quadratic programming subproblem at each iteration, ultimately converging to a locally optimal solution.

In our setting, selecting the operating point for a fixed order σ_f can be done by applying the SQP-algorithm to the Lagrangian function of the optimization problem (7)

$$\mathcal{L}(\mathbf{x}, \nu_1, \nu_2) = U(\sigma_f, \mathbf{x}) - \nu_1^T(\mathbf{x} - 1) + \nu_2^T \mathbf{x}.$$

Because of the gradient-descent nature of the SQP algorithm, it is not possible to guarantee convergence to the global maximum and the convergence may only be locally optimal. However, the SQP algorithm can be initialized with multiple starting configurations in order to find a better local optimum (or even the global optimum). Since the number and size of local optima depend on the shape of the various DET curves of each classifier, a rigorous bound on the probability to find the global optimum cannot be proven. However, certain start regions are more likely to converge to better local optimum.[2]

SQP-Greedy Algorithm for Joint Ordering and Operating Point Selection: Given this SQP-based solution for operating point selection, we can combine it with our iterative greedy order selection to build a joint order and operating point selection strategy. This iterative approach is summarized as follows.

Centralized Algorithm 2 SQP-Greedy algorithm

- **Initialize** $\sigma^{(0)}$.
- **Repeat** until Greedy Algorithm does not modify order.
  1. Given order $\sigma^{(j)}$, compute locally optimal $\mathbf{x}^{(j)}$ through SQP
  2. Given operating points $\mathbf{x}^{(j)}$, update order $\sigma^{(j+1)}$ using (A-)Greedy algorithm.

Each step of the SQP-Greedy algorithm is guaranteed to improve the global utility of the problem. Given a maximum bounded utility, the algorithm is then guaranteed to converge. However, it is not possible to bound the performance gap between the SQP-Greedy and the optimal algorithm with a constant factor, since the SQP only achieves the local maximum.

V. Analysis for Multi-Query Settings

A. Simultaneous Processing of Multiple Queries

Until now, we concentrated our efforts on the processing of one single query q and considered the problem of selecting the order and configuration of the set of classifiers $\mathcal{C}(q) = \{C_1, \ldots, C_N\}$ to achieve the best performance in terms of accuracy and delay. The methodology proposed could be extended to the identification and processing of data streams belonging to different queries $\{q_1, \ldots, q_M\} \in \mathcal{Q}$ by performing Greedy Algorithms in parallel for each query q_j.

However, a stream mining system must not only be seen as a query-centric processing system aiming to identify which subset of data answers a given set of queries. Instead of defining the set of classifiers on the basis of the set of queries $(\mathcal{C} = \bigcap_{q \in \mathcal{Q}} C(q))$, we can determine what are all the queries which can be answered given a set of classifiers $(\mathcal{Q} = \{q | \mathcal{C}(q) \subset \mathcal{C}\})$. Indeed, classifier design is often expensive and feature extraction by each classifier should therefore be leveraged as much as possible. Hence, the set of available classifiers should be reused across as many queries as possible.

Such classifier-centric approach leads to an explosion of the number of queries that can be processed: a set of N binary classifiers can potentially process $\sum_{k=1}^{N} \binom{N}{k} 2^k$ different queries. As a consequence, we must design ordering and configuration selection algorithms capable of processing simultaneously data stream belonging to a large set of queries $\mathcal{Q}$.

[2]For example, since the operating point $p^F = 0$ corresponds to a saddle point of the utility function, it would achieve steepest utility slope. Furthermore, the slope of the DET curve is maximal at $p^F = 0$ (due to concavity of the DET curve), such that high detection probabilities can be obtained under low false alarm probabilities near the origin.

B. Requirement for Algorithms to Meet Time Constraints

We mentioned in Section IV-C the need for algorithms to take into account the system 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. Such plurality of queries can lead to abrupt changes over small intervals of time: two consecutive tuples could belong to two different 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 require rapid adaptation of the order and operating points, often even at the granularity of one tuple. Hence, any optimization algorithm 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 $\tau \leq \mathfrak{C}\tau^{\mathrm{dyn}}$ where τ^{dyn} represents the characteristic time of significant change, and $\mathfrak{C}$ represents a safety factor to account for smoothing of burstiness in data volume, and dynamic variations in resource availability that affect the time taken to process individual tuples.

C. System Constraints for Centralized Approach

The centralized approach exposed in Section IV imposes several system constraints.

- **System and Information Constraint**: There needs to be a central agent that collects all information, generates optimal order and operating points for each classifier, and distributes and enforces results on all classifiers. This creates a bottleneck, as well as a single point of failure.
- **Query Specificity**: The order and operating points of classifiers depend on the nature of the query of interest: query q requires a specific set of classifiers $\mathcal{C}(q)$, and has specific utility parameters c^M, c^F and λ.
- **Sensitivity to Dynamics**: Centralized approaches are inefficient under highly dynamic environments. Indeed, the solution provided by the algorithm in one iteration is obsolete if the time required by the central agent to compute the optimal order and operating points (e.g., for SQP-Greedy) is greater than the characteristic time of the environment dynamics (i.e., stationarity holds on average for less than this time): $\tau \leq \mathfrak{C}\tau^{\mathrm{dyn}}$.

Two approaches, exposed in Fig. 2, could be envisaged to determine the optimal order and operating point of a system with M different queries being performed simultaneously. A first solution would consist in running the SQP-Greedy Algorithm for each query. This solution, adequate for static system characteristics, would however not be efficient in dynamic settings. Indeed, since the computational requirement and the number of message exchanges for the optimization increase proportionally to the number of queries, the time $M\tau$ required to compute a solutions for each query would greatly increase and the solution would arrive too late, i.e., $M\tau > \mathfrak{C}\tau^{\mathrm{dyn}}$ for large M. Another option for determining the optimal order and operating point of the stream mining system would be to solve the optimization problem for the sum of weighted utilities across all queries.

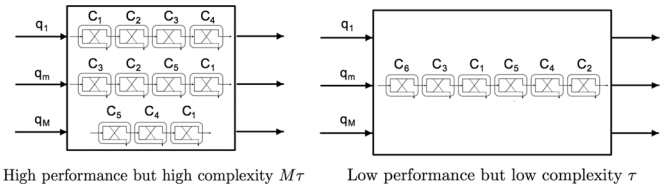


Fig. 2. Centralized processing of multiple queries.

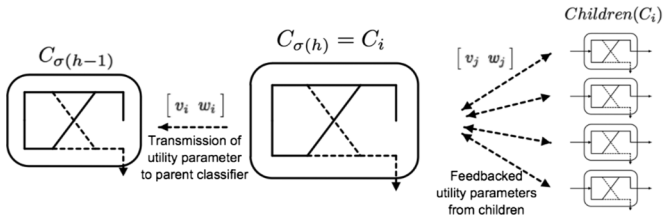


Fig. 3. Feedbacked information for decentralized algorithms.

However, this is likely to result in lower efficiency, since there would only be a single common order for all queries.

In order to answer these issues, we propose a decentralized approach which will simultaneously increase performance of stream mining system, better comply with changing dynamics in a broad sense and provide query-specific orders, while being much more adaptable and "customizable" to specific settings.

Given a distributed setting, and processing variability, the processing time α of classifiers may vary from one classifier to another. As a result, transmission from one classifier to another is asynchronous. We will refer to the local time of classifiers C_i using the subscript $\mathfrak{t}_i$ in reference to its local clock. Nonetheless, as we will discuss in Section VI-B, we can implement decentralized asynchronous algorithms that converge under certain stationarity assumptions.

VI. Decentralized Approach

A. Decentralized Optimization Problem

As described in Section V-B, stream tuple characteristics can vary significantly from one tuple to another. Hence, it may be required to determine the best order per data tuple. The decentralized algorithm proposed in the following section models the topology construction as a dynamic routing problem, which is then performed tuple by tuple. The key idea of the decentralized algorithm is to replace centralized order selection by local routing decisions, i.e., determining which classifier to forward the stream to. To describe this, we set up a Markovian cooperative common interest game framework $\{\mathcal{C}, \mathcal{S}, \mathcal{A}, \mathcal{U}\}$ [12], where:

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

Users of the Stream Mining System: Consider N classifiers $\mathcal{C} = \{C_1, \ldots, C_N\}$. The classifiers are supposed to be autonomous: unless otherwise mentioned, they do not communicate with each other and they take decisions independently. We recall that the hth classifier will be referred as $C_i = C_{\sigma(h)}$. We will also refer to the stream source as $C_0 = C_{\sigma(0)}$.

States Observed by Each Classifier: The set of states can be written as $\mathcal{S} = \times_{i \leq N} \mathcal{S}_i$, where $\mathcal{S}_i$ is the local state set of

classifier $C_i = C_{\sigma(h)}$ at the hth position in the classifier chain. $\mathcal{S}_i = \{(Children(C_i), \theta_i)\}$, with $Children(C_i) = \{C_k \in \mathcal{C} | C_k \notin \{C_{\sigma(1)}, \ldots, C_i\}\} \subset \mathcal{C}$, and $\theta_i = (t^\sigma_{h-1}/g^\sigma_{h-1}) \in [1 \infty]$.

$Children(C_i)$ represents the subset of classifiers through which the stream still needs to be processed after it passes classifier C_i. This is required identification information that can be included (as an overhead) along with each stream tuple so that it is not sent to a classifier that has already processed it. The estimated throughput-to-goodput ratio θ_i is the ratio between the throughput and the goodput of the stream entering classifier C_i. It is a measure of the accuracy of the ordered set of classifiers $\{C_{\sigma(1)}, C_{\sigma(2)}, \ldots, C_i\}$. Indeed, $\theta_i = 1$ corresponds to perfect classifiers $C_{\sigma(1)}, C_{\sigma(2)}, \ldots, C_i$, (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 classifier $C_i = C_{\sigma(h)}$ can observe t^σ_{h-1}, it is equivalent to send g^σ_{h-1}. Notice that this goodput value could also be computed recursively if knowledge on classifiers is available.[3]

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 $B_l = [b_{l-1}\ b_l]$ 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 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: $\forall h \in [1\ N], C_{\sigma(h)} = C_{\sigma(h-1)\rightarrow}$. This corresponds to solving a dynamic routing problem to construct the topology in a distributed manner.

Local Utility of a Classifier: We define the local utility of a chain of classifiers by backward induction

$$U_{\sigma(h)} = -\rho_{\sigma(h)} t^\sigma_{h-1} + U_{\sigma(h+1)} \tag{9}$$

with $U_{\sigma(N)} = -\rho_{\sigma(N)} t^\sigma_{N-1} + g^\sigma_N - K t^\sigma_N$. The end-to-end utility of the chain of classifiers can then be reduced to $U = U_{\sigma(1)}$.

Proposition 1 (Local Utility): Each classifier $C_i = C_{\sigma(h)}$ will globally maximize the system's utility by autonomously maximizing its local utility $U_i = \underbrace{[v^\sigma_h\ w^\sigma_h]}_{=[v_i\ w_i]} \begin{bmatrix} t^\sigma_{h-1} \\ g^\sigma_{h-1} \end{bmatrix}$ where the local utility parameters $[v^\sigma_h\ w^\sigma_h]$ are defined recursively

$$\begin{aligned} [v^\sigma_N \quad w^\sigma_N] &= -\begin{bmatrix} \rho_{\sigma(N)} & 0 \end{bmatrix} + [-K \quad 1] T^\sigma_N \\ [v^\sigma_i \quad w^\sigma_i] &= -[\rho_i \quad 0] + \begin{bmatrix} v^\sigma_{i+1} & w^\sigma_{i+1} \end{bmatrix} T^\sigma_i \end{aligned}$$

Proof: cf. Appendix A, Result no 4. ■

Markovian Property of the Game: The local utility of classifier C_i can be rewritten as

$$U_i = \left(-[\rho_i \quad 0] + \begin{bmatrix} v^\sigma_{h+1} & w^\sigma_{h+1} \end{bmatrix} T_i(x_i)\right) \begin{bmatrix} t^\sigma_{h-1} \\ g^\sigma_{h-1} \end{bmatrix} \tag{10}$$

and the optimization may be rewritten as

$$(\sigma^*, x^*) = \underset{\substack{\sigma \in Perm(N) \\ x_i \in [0,1]}}{\arg\max} \frac{U_i}{g_i}.$$

Therefore, the decision of classifier C_i only depends on its operating point x_i, on the state θ_i which it observes[4] and on the local utility parameters $[v_j\ w_j]$ of its children classifiers $C_j \in Children(C_i)$. The process of backward induction of these local utility parameters, using feedback from children classifiers, is shown in Fig. 3. Being provided with the utility parameters of all its 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.

This indicates that states at time $\mathfrak{t}_i + 1$ only depend on the actions taken at time $\mathfrak{t}_i$. Indeed, denoting $\sigma^{(\mathfrak{t}_i)}$ as the order of classifiers at time $\mathfrak{t}_i$, we have $Children(C_i)^{(\mathfrak{t}_i+1)} = \{C_k \notin \{C_{\sigma^{(\mathfrak{t}_i)}(1)}, C_{\sigma^{(\mathfrak{t}_i)}(2)}, \ldots, C_i\}$, and $\theta_i^{(\mathfrak{t}_i+1)} = t^{\sigma^{(\mathfrak{t}_i+1)}}_{h-1} / g^{\sigma^{(\mathfrak{t}_i+1)}}_{h-1}$, where $\begin{bmatrix} t^{\sigma^{(\mathfrak{t}_i+1)}}_{h-1} \\ g^{\sigma^{(\mathfrak{t}_i+1)}}_{h-1} \end{bmatrix} = \prod_{k=1}^{h-1} [T^{\sigma^{(\mathfrak{t}_i)}}_k (x^{(\mathfrak{t}_i)}_{\sigma^{(\mathfrak{t}_i)}(k)})] \begin{bmatrix} t_0 \\ g_0 \end{bmatrix}$.

B. Decentralized Algorithms

1) Distributed Ordering: 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.

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 $[v_j\ w_j]$. Since the probed child requires knowledge of its state to determine its utility parameter corresponding to its optimal decision, the parent classifier must send it both the set of remaining children $Children(C_i)$ and its throughput-to-goodput ratio θ_i. Observe that these messages exchanges are very limited in size.

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 computer their local utility, each of them then probes its children for their utility parameters $[v\ w]$. 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

[3]Indeed, supposing a well-classified input data: $t_0 = g_0$, using notation $T_i = \begin{bmatrix} a_i & b_i \\ 0 & c_i \end{bmatrix}$, the relationship $t_i/g_i = a_i t_{i-1} + b_i g_{i-1}/c_i g_{i-1} = (a_i/c_i)(t_{i-1}/g_{i-1}) + (b_i/c_i)$ leads to $\theta_i = t_{i-1}/g_{i-1} = \prod_{j=1}^{i-1}(a_j/c_j) + \sum_{j=1}^{i-1}(b_j/c_j)\prod_{k=1}^{j-1}(a_k/c_k)$.

[4]t_{i-1} and g_{i-1} are not required since: $\arg\max U_i = \arg\max(U_i/g_{i-1}) = (-[\rho_i\ 0] + [v_{i+1}\ w_{i+1}]T_i)\begin{bmatrix} \theta_i \\ 1 \end{bmatrix}$.

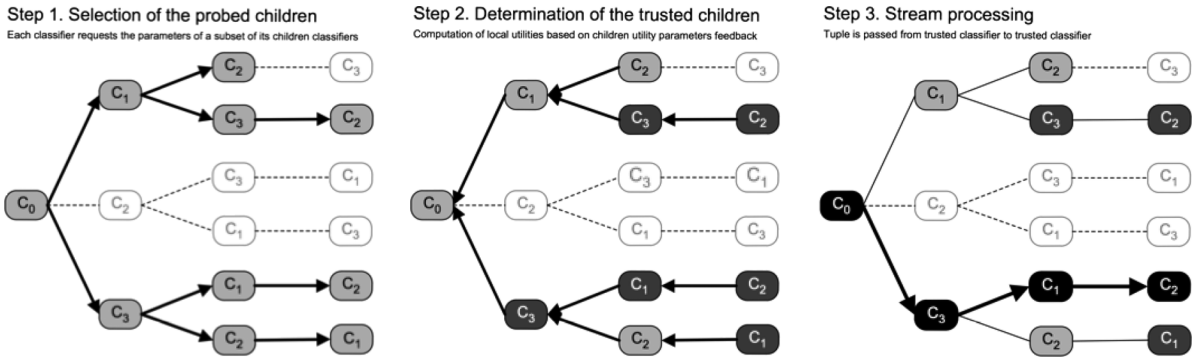


Fig. 4. Global Partial Search Algorithm.

classifiers to the root classifier C_0. The order yielding the maximal utility is selected.

Observe that this decentralized ordering algorithm leads to a full exploration of all $N!$ possible orders at each iteration. Achieving the optimal order only requires one iteration, however this iteration consists requires $O(N!)$ operations and may thus take substantial time.[5] 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 optimal order calculated by the algorithm would not arrive too late and thus be completely obsolete. In particular, the time constraint $\tau \leq \mathfrak{C}\tau^{\mathrm{dyn}}$, underlined in Section V-B, must not be violated.

We therefore need algorithms capable of determining quickly 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.

Global Description of the Partial Search Ordering Algorithm: The key insight of the Partial Search Algorithm is to probe only a selected subset of the $N!$ orders at each iteration. In other words, instead of probing all its children classifiers systematically, the hth classifier only probes a subset of its $N - h$ children.

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

Step 1) **Selection of the children to probe**. A partial tree is selected recursively (light gray 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, and at each node of the tree, the child classifier which provides its parent classifier with the greatest local utility is selected as trusted child (dark gray in Fig. 4).

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

[5]This can take $\tau > 5$ minutes for seven classifiers, as will be shown in Section VIII-B.

If we want to describe Step 1 more specifically, classifier C_i will probe its child C_j with probability p_j^i. As will be shown in Section VII, adjusting the values of p_j^i will enable the classifiers to adapt the number of operations and the time τ required per iteration: indeed, for low values of p_j^i, few of the $N!$ orders will be explored; since each classifier only probes a small fraction of its children, one iteration will be very rapid. However, if the values of p_j^i are close to 1, each iteration requires a substantial amount of probing and one iteration will be long.

Local Description of the Partial Search Ordering Algorithm: Observe that one classifier may appear at multiple depth and position in the classifiers' tree. Each time, it will realize a local algorithm described in the following flowchart.

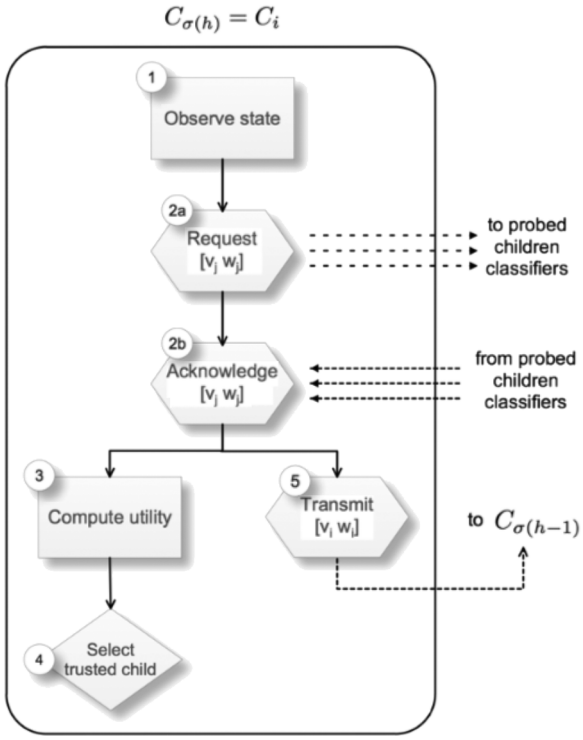
Decentralized Algorithm 3 Partial Search Ordering Algorithm—*for classifier* $C_i = C_{\sigma(h)}$

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

2) Distributed Ordering and Operating Point Selection: 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.[6] 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

$$U_i(C_j) = \max_{x_i} (-[\rho_i \quad 0] + [v_j \quad w_j]T_i(x_i)) \begin{bmatrix} t_j \\ g_j \end{bmatrix}. \qquad (11)$$

[6]The utility parameters $[v_j\ w_j]$ feedbacked from classifier C_j to classifier C_i are independent of any classifiers' operating points, through their determination by classifier C_j required to optimally selecting its operating point x_j.



To solve the local optimization problem (11), each classifier can either use gradient descent 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), as in [21].

3) Robustness of the Partial Search Ordering Algorithm: Message Exchanges: The Partial Search Algorithm requires message exchange: a probed classifier must be sent the remaining set of classifiers and the throughput-to-goodput ratio of its parent classifier and will send back its utility parameters. If the amount of per tuple messages that need to be exchanged exceeds the system's capacity (which will especially be the case if the depth of the tree is big), three solutions can be implemented. 1) The optimization may be performed on a per-group of tuples basis (which will be routed together), instead of for one tuple. This will reduce the overhead. 2) We can adapt the amount of probing by adjusting the parameter p. This will however reduce the speed of convergence. 3) Each classifier can keep a memory of utility from past experiments: this will be emphasized in Section VII-C.

Proposition 2 (Convergence of Partial Search Algorithm): Under stable conditions the Partial Search Algorithm converges and the equilibrium point corresponds to a Nash equilibrium of the Markovian game.

For fixed operating point the Partial Search Algorithm converges to the optimal order if $p_j^i > 0 \forall i, j$.

Proof: cf. Appendix A, Result no 5. The main idea for proving convergence is that the utility is by construction non-decreasing. Indeed if a new probed route provides a lower utility than the previously used route, it is this previous route which will be kept and enforced (and the utility will remain the same for this iteration).

In case of joint ordering and operating point selection, there exist multiple Nash Equilibria, each corresponding to a local minimum of the utility function. The selection of the Nash Equilibrium among the set of possible Nash Equilibria depends on the initial condition (i.e., order and operating points) of the algorithm. To select the best Nash Equilibrium, we can perform the Partial Search Algorithm for multiple initial conditions and keep only the solution which yielded the maximum utility.

Convergence Speed: In practice, the stable input conditions 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.

In analogy to [8], we first discuss how payoff-based Safe Experimentation, a heuristic case of Partial Search Algorithm can be used for decentralized stream mining and leads to a low-complexity algorithm, however with slow convergence rate. Fortunately, the convergence speed of the Partial Search Algorithm can be improved by appropriately selecting the probing probabilities p_j^i. In Section VII, we will construct a model-based algorithm which enables to control the convergence properties of the Partial Search Algorithm, and lead to faster convergence.

C. Safe Experimentation

We will benchmark our results against Safe Experimentation algorithms as cited in [8]. This low-complexity, payoff-based learning approach was first proposed for large-scale, distributed, multi-agent systems, where each agent is unable to observe the actions of all other agents (due to informational or complexity constraints) and hence cannot build a model of other agents [18]. The agent therefore adheres to a trusted action at most times, but occasionally explores a different one in search of a potentially better action.

Safe Experimentation is a reinforcement learning algorithm where each classifier learns by observing the payoff with which its past actions were rewarded. As such, it does not consider the interactions between agents, or in our case, the actions of other autonomous classifiers. In particular, there is no explicit message exchange among classifiers required (i.e., no $[v\ w]$ exchanged), though each classifier needs to know the reward of its action.

Safe Experimentation Algorithm: The Safe Experimentation algorithm is initialized by selecting an initial order σ_0 of classifier. $C_{\sigma_0(h+1)}$ will be referred as the "trusted" child of the hth classifier $C_{\sigma_0(h)}$. At each time slot, $C_{\sigma(h)}$ will either forward the stream to its "trusted" child $C_{\sigma(h+1)}$ with probability $(1-\varepsilon)$ or, with probability ε, will explore a classifier C_j chosen randomly among its children. In the case where a higher reward is achieved through exploration, C_j will become the new "trusted" child of $C_{\sigma(h)}$.

The name of Safe Experimentation comes from the fact that the global utility achieved by the algorithm is non-decreasing over time. Since utilities are upper-bounded, this ensures convergence of the Safe Experimentation algorithm. Furthermore, so long as $\epsilon > 0$, all possible orders will ultimately be explored, such that Safe Experimentation converges to the optimal order [8].

Instead of considering a fixed exploration rate ϵ, we can consider a diminishing exploration rate $\epsilon_{\mathfrak{t}}$. In this way, the algorithm will explore largely for first iterations and focus on exploited orders near convergence. $\epsilon_{\mathfrak{t}} \to 0$ and $\prod_{\tau=1}^{t}(1-(\epsilon_\tau^{N-1}/(N-1)!)) \to 0$ are sufficient conditions for convergence, typically verified for $\epsilon_{\mathfrak{t}} = (1/\mathfrak{t})^{1/n}$.

Limits of Safe Experimentation: **Slow convergence**: One iteration of Safe Experimentation is very rapid $(O(N))$, since only one order is experienced. However, the expected number of iterations required to converge to optimal order is bounded below by $N!$ (corresponding to uniform search: $\epsilon_t = 1$.) As a consequence, the time required to reach the optimal solution might be infinitely long, since the optimal order could be experimented after an infinitely large number of iterations.

General approach: The slow convergence of Safe Experimentation is due to the fact that it does not leverage the problem structure, and supports probing (i.e., requesting the utility parameters) only one child classifier at each stage of the search. In general, the ability to probe multiple child classifiers simultaneously and selecting from among them is likely to increase the rate of convergence. In the following section, we build a parameterized approach that can improve the convergence rate through the appropriate selection of a probing probability p_j^i.

VII. Parametric Partial Search Order and Operating Point Selection Algorithm

In this section, we construct an algorithm that trades off convergence rate with the utility (7) by appropriately determining the search probabilities p_j^i of the Partial Search Algorithm. 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 $[v_j\ w_j]$. Performing an experiment can lead to a higher utility, but will induce a cost in terms of computational time.

- Denote by $\hat{U}(E_{i\to j}|s_k)$ 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. This computational time will be assumed independent of the classifiers involved in the experiment performed and the state observed.

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 probing probabilities p_j^i then need to be determined to maximize the total expected utility within a certain time

$$\begin{cases} \underset{p_j^i\in[0\ 1]}{\text{maximize}} & \hat{U}\left(p_j^i\right) \\ \text{subject to} & \tau\left(p_i^j\right) \le \mathfrak{C}\tau^{\text{dyn}} \end{cases}. \tag{12}$$

We consider three important tradeoffs that control the search: 1) how much to search? 2) how deep to search? 3) where to search? Based on these, we construct a parametric learning algorithm. We will show that the probability that the hth classifier $C_{\sigma(h)} = C_i$ probes its children C_j, given that it received data with throughput-to-goodput ratio $\theta_i \in S_k$ can be expressed in terms of control parameters (p, ξ, β) as

$$p_j^i = \underbrace{p}_{\substack{\text{how much}\\ \downarrow \\ \text{cf. Section VII-A}}} \times \underbrace{\frac{C'}{1+e^{-\frac{h-\lfloor Np\rfloor}{\xi}}}}_{\substack{\text{how deep?}\\ \downarrow \\ \text{cf. Section VII-B}}} \times \underbrace{\frac{e^{\beta U_i(j,k)}}{\sum_{C_l\in\text{Children}(C_i)} e^{\beta U_i(l,k)}}}_{\substack{\text{where?}\\ \downarrow \\ \text{cf. Section VII-C}}}$$

A. How Much to Search—Efficiency Versus Flexibility

In this section, we determine the average probability of search $p = Mean(p_j^i)\forall i,j$. This parameter p is used to arbitrate between rapid but inflexible search and slower but system-compliant search. Its value will impact the time τ required per iteration and it needs to be selected small enough in order to ensure that $\tau \le \mathfrak{C}\tau^{\text{dyn}}$.

Performance of Algorithm for Random Search: Consider a random search: any classifier C_i probes its children with the same probability $p_j^i = p$. $p \approx 0$ corresponds to algorithms which seldom perform new search and probe only a few branches of the search 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 probe nearly all the branches of the search tree at each iteration. Hence, reaching the optimal order requires on average $1/p^N \approx 1$ slow iterations, each with complexity $O(N!p^N \approx O(N!))$.

For a given p, the average number of children that classifier $C_{\sigma(h)}$ probes is

$$n_h = \sum_{k=0}^{N-k} k \underbrace{\binom{k}{N-h} p^k (1-p)^{N-h-k}}_{\mathbb{P}(k\ children\ probed)} = (N-h)p.$$

The average number of orders explored per iteration is thus $N_O = \prod_{h=0}^{N-1} n_h = N!p^N$ while the processing time is $\tau(p) = \tau^{ex}\sum_{h=0}^{N-1}(\prod_{i=0}^{h} n_i) = \tau^{ex}\sum_{h=0}^{N-1}(N!/(N-h)!)p^h$, which is increasing on [0 1].

Speed of Search: Since the utility $\hat{U}$ expected from a set of experiments $(E_{i\to j})$ cannot be determined analytically, we focus on optimizing the speed of search V, defined as the average number of orders explored in a certain amount of time. $V(p) = (N_O/\tau) = (\tau^{ex}\sum_{h=1}^{N}(1/(N-h)!p^{N-h}))^{-1}$ This choice corresponds to the assumption that expected utility is proportional to the speed of search. Hence, the optimization problem in (12) becomes

$$\begin{cases} \underset{p\in[0\ 1]}{\text{maximize}} & V(p) \\ \text{subject to} & \tau(p) \le \mathfrak{C}\tau^{\text{dyn}} \end{cases} \qquad \mathfrak{C} \ll 1. \tag{13}$$

Since both the objective function and the constraint function are increasing in p, the optimal average probing p^* will be obtained by saturating the time constraint: $\tau(p) = \mathfrak{C}\tau^{\mathrm{dyn}}$. Hence, the optimal average probe probability for random search may be determined as $p^* = \tau^{-1}(\mathfrak{C}\tau^{\mathrm{dyn}})$. Note that a constant probe probability ignores the different impact of each classifier on the end to end utility. In the following sections, we determine probe probabilities as a function of the depth of the classifier in the tree.

B. How Deep to Search—Speed Versus Exhaustiveness

In this section, we examine the tradeoff between speed of convergence and exhaustiveness of search. We start by allowing classifiers to have different probe probabilities depending on their depth in the chain (correspondingly search tree). Suppose that the hth classifier $C_{\sigma(h)}$ searches through all its children C_j with the same probability $p_j^{\sigma(h)} = p^{\sigma(h)}$. We want to construct a weight vector $\mathbf{d} \in [0\,(1/p)]^N$ such that $p^{\sigma(h)} = p\,d_h$, where p represents the random search probability introduced previously. d_h, thus controls how the probe probability of the hth classifier $p^{\sigma(h)}$ deviates from the average value p. Under the condition that p is the average value of all $p^{\sigma(h)}$, we have $\sum_{h=1}^{N} d_h = N$.

We can show that the average number of children probed by classifier $C_{\sigma(h)}$ then becomes $n_h = (N-h)pd_h$. Then the iteration processing time is given by

$$\tau(p,d) = \tau^{ex} \sum_{h=0}^{N-1} \left(\frac{N!}{(N-h)!} p^h \prod_{g=1}^{h} d_g \right)$$

. As proved in Appendix A, Result no 6, the weight d_h^* which minimizes the processing time τ for fixed p may be expressed as

$$d_h^* = \begin{cases} 0, & \text{if } h < H \\ d_H, & \text{if } h = H \\ 1/p, & \text{if } h > H \end{cases}$$

where $H = N - 1 - \lfloor Np \rfloor$ and $d_H = N - \lfloor Np \rfloor / p$. This basically translates into increasing amount of search at increasing depths of the tree.

The corresponding probing probability is $pd^* = (p^{\sigma(1)}, \ldots, p^{\sigma(N)}) = (\underbrace{0, \ldots, 0}_{N-1-\lfloor Np \rfloor}, p_H, \underbrace{1, \ldots, 1}_{\lfloor Np \rfloor})$. Therefore, if we use these probing probabilities, the first $\lfloor Np \rfloor$ classifiers will remain unchanged. As a consequence, the Partial Search Algorithm will not converge to the optimal solution, since some orders will never be explored. To avoid this scenario, ensure a minimal quality of service and impose that the search is exhaustive and considers all orders, we need to approximate the weight vector d^* by a "smoother" vector with strictly positive elements. Plotting $h \mapsto d_h^*$ reveals a quasi-threshold function. We choose to approximate this curve by a sigmoid $x \mapsto (1/1 + e^{-(h-H/\xi)})$. The corresponding approximation of d_h^* can then be expressed as

$$\tilde{d_h} = \frac{C_\xi}{1 + e^{-\frac{h - \lfloor Np \rfloor}{\xi}}} \tag{14}$$

where C_ξ is a normalization coefficient such that $\sum_{h=1}^{N} \tilde{d_h} = N$. The control parameter ξ is a refinement parameter, which dictates how much more extensive search should be performed with increasing depth. As ξ goes from 0 to ∞, it corresponds to searching only higher depths (as without smoothing) to searching all depths uniformly.

TABLE I
CLASSIFIER CHARACTERISTICS

Classifier	Delay coef. α	Accuracy coef. AUG
C_1	.25	.82
C_2	.5	.87
C_3	.75	.93
C_4	1	.98

C. Where to Search—Exploration Versus Exploitation

In this section, we discuss the tradeoff between exploitation of tested actions (previously probed children) and exploration of new actions (probing new children). Until now, we focused on the number of orders explored. However, the real variable of interest is the number of *new* orders explored.

Learning From Prior Experiments: Given an average search probability $p^{\sigma(h)}$, we can encourage probing unexplored children or exploit already-visited orders, by weighting the propensity of classifier $C_i = C_{\sigma(h)}$ to probe one of its children classifier C_j, based on reward from previous experiments.

- The algorithm can *exploit* the available orders to select the latest best—known—order by allocating large values of $p_j^{\sigma(h)}$ to children that provided high utility in the past.
- The algorithm can *explore* new orders by allocating large values of $p_j^{\sigma(h)}$ to non-visited children or children which brought lower local utility.

Unlike traditional learning scenarios [10], our algorithm can base its exploration versus exploitation decision based on immediate feedback in terms of the achieved utility—hence, exploration does not hurt performance, as we can test orders without selecting them.

State-Based Learning Algorithm: At this stage, the hth classifier $C_{\sigma(h)} = C_i$ probes any of its children with a probability $p^{\sigma(h)}$ independent on the child considered. Yet, forwarding the stream to some of these children might have resulted, in the past, in higher local utility, rather than forwarding the stream to some other children. To monitor past local experiments' impact on its utility, classifier C_i needs to keep record of the latest local utility $U_i(j)$ it achieved when it transmitted the stream to classifier C_j.

Since the choice of orders depends on the throughput-to-goodput ratio θ_i observed by each classifier, we can refine learning by keeping track of the latest utility $U_i(j,k)$ achieved by classifier C_i when it performed experiment $E_{i \to j}$ with a stream with $\theta_i \in S_k$. As before, we use a discretized state space using nonuniform quantization techniques (with L bins) for reduced storage and computational requirements. We can thus define a three-step algorithm as follows.

Step 1: Children selection. If the throughput-to-goodput ratio for classifier $C_i = C_{\sigma(h)}$ falls into the kth bin: $\theta_i \in S_k$, C_i will probe its child $C_j \in Children(C_i)$ with probability $p_j^i = p^{\sigma(h)}(e^{\beta U_i(j,k)} / \sum_{C_l \in Children(C_i)} e^{\beta U_i(l,k)})$.

TABLE II
COMPARISON OF ORDERING ALGORITHMS

Ordering Algorithm	System compliance	Utility achieved	Message exchange	Speed of convergence	Adaptability	Control
No algorithm	∅	Suboptimal	∅	∅	∅	∅
Adaptive-Greedy	Low	Bounded	Heavy	Very rapid	Little	∅
Safe Experimentation	High	Optimal	∅	Medium	∅	∅
Partial Search (proposed)	Complete	Optimal	Light	Rapid	Total	Yes

Step 2: Utility update. Denote by $\{C_{j_1}, \dots C_{j_m}\} \subset Children(C_i)$ the probed children. Based on their parameters, the utility matrix is updated as $U_i(j_l, k) = \max_x([\rho_i\ 0] + [v_{j_l}\ w_{j_l}]T_i(x)) \begin{bmatrix} t^{\sigma}_{h-1} \\ g^{\sigma}_{h-1} \end{bmatrix}, \forall l \leq m.$

Step 3: Stream processing. The stream is then forwarded to the child which that results in the highest utility.

In practice, the weight assigned to a child classifier based on its past reward can be determined using any increasing function γ. Using $\gamma_\beta(U) = (e^{\beta U}/\sum_V e^{\beta V})$ is motivated by the analogy of a classifiers utility U to an energy [23]. In this case, $e^{\beta U}/\sum_V e^{\beta V}$ 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. As β goes from 0 to ∞, it corresponds to going from random exploration to full exploitation.

D. Comparison of Proposed Algorithms' Behaviors

As a summary, we propose the following analysis of algorithms. We benchmark our result on random order and configuration selection. We then compare the Greedy Algorithm (cf. Section IV-A), Safe Experimentation (cf. Section VI-C) and our proposed Partial Search Algorithm (cf. Section VII).

Table VII-D and VII-D compare the performance of the ordering algorithms and joint ordering and operating point algorithms based on essential criteria.

VIII. EXPERIMENTAL RESULTS

Our experiments were performed in Matlab on a MacBook with 2.4-GHz Intel Core 2 Duo Processor and 2-GB 667 MHz DDR2 SDRAM. The following experimental results illustrate the three major tradeoffs presented in our discussion. In a first part, we show the impact of cost parameters c^M, c^F, and λ on the order and operating point selected. We then compare the performance and the time required to perform the different algorithms presented in this paper. Finally, we discuss the learning tradeoffs concerning the decentralized algorithm.

A. Accuracy Versus Delay Tradeoff Analysis

Upper Bound of Delay Cost $\lambda_{\max}$: Given that the objective function of the stream mining optimization problem (6) is a tradeoff between misdetection, misclassification, and processing delay, we can expect the order and operating point selected to vary depending on the values of the tradeoff parameters c^M, c^F, and λ.

Since we can bound the utility U_λ by $-\min(\alpha_k)t_0 < U_\lambda < (1 - K - \sum \rho_i)\Phi$, we can show (cf. Appendix A, Result no 7.) that for $\lambda > \lambda_{\max} = c^M/\sum_{k=1}^N \alpha_k - \min(\alpha_k)t_0/\Phi$, 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 $C_1, \dots, C_4$ with different processing costs α_i and different DET curves, characterized by a specific area under graph (AUG). We quantize the DET curve and keep four operating points $x_1 = p^F_1 = 0$, $x_2 = p^F_2 = 0.33$, $x_3 = p^F_3 = 0.66$, and $x_4 = p^F_4 = 1$. For this experiment, we use classifiers with increasing α and decreasing AUG, as shown in Table I, 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. Indeed, since for high relative value of λ the delay cost should be minimized, the classifiers with the lowest processing time should be processed first. In this way, classifiers with highest processing time will only need to process a fraction of the stream, corresponding to the tuples which have not been dropped by the previous classifiers. At the opposite, for low relative value of λ, classifiers with highest accuracy (i.e., highest AUG) should be positioned first, in order to limit error propagation.

In Tables II–IV, we show how both the optimal order of classifiers and the optimal operating points individually selected change for varying tradeoff parameters c^F, λ, and fixed $c^M = 10$. The results are obtained by computing the utility achieved for every order and operating point and keeping the optimal setup.

If delay is not taken into account ($\lambda = 0$), classifiers are ordered with decreasing accuracy: indeed error propagation should be avoided. However, for $\lambda = \lambda_{\max}/30$ and $\lambda = \lambda_{\max}/10$, we verify that classifiers with lowest transmission cost α are used first. Furthermore, we observe that for $\lambda = \lambda_{\max}/10$, the delay cost becomes too important and the stream is not processed (this means that the optimal policy is for the classifier C_1, with lowest processing cost α, to drop all tuples).

The condition $c^F = 0$ and $\lambda > 0$ corresponds to a tradeoff situation, where classifiers are ordered according to both their accuracy and their transmission cost. The first classifier is selected on the basis of its delay, with a low processing time. However, the last three classifiers are chosen based on their accuracy to limit error propagation. One explanation to this would be that since throughput are rapidly decreasing ($t_i \approx \phi t_{i-1}$), the delay penalty $c_{delay} = \lambda \sum_{i=0}^{3} \alpha_{\sigma(i)} t_{i-1}$ can be approximated to $\lambda \alpha_{\sigma(1)} t_0$. However, the misclassification penalty $c_{err} = c^M(\Phi t_0 - g_N) + c^F(t^{\sigma}_N - g_N)$ only depends on the final throughput and goodput. Therefore, selecting a classifier with high processing time α will lead, as a first approximation, to a high delay cost, while the impact of selecting a classifier with low precision would have a more limited impact. However,

TABLE III
COMPARISON OF ORDERING AND OPERATING POINT SELECTION ALGORITHMS

Ordering and Operating Point Selection Algorithm	System compliance	Utility achieved	Message exchange	Speed of convergence	Adaptability	Control
No algorithm	∅	Suboptimal	∅	∅	∅	∅
SQP-Greedy	Low	Bound ; local opt.	Heavy	Medium	Little	∅
Safe Experimentation	High	Local optimum	∅	Medium	∅	∅
Partial Search (proposed)	Complete	Local optimum	Light	Rapid	Total	Yes

TABLE IV
OPTIMAL ORDER AND OPERATING POINT FOR VARYING OBJECTIVE PARAMETERS c^M AND λ

		$c^F=0$	$c^F=1$	$c^F=2$	$c^F=5$
$\lambda=0$	σ	$[C_4\ C_3\ C_2\ C_1]$	$[C_4\ C_3\ C_2\ C_1]$	$[C_4\ C_3\ C_2\ C_1]$	$[C_4\ C_3\ C_2\ C_1]$
	$\mathbf{x}$	$[x_4\ x_4\ x_4\ x_4]$	$[x_4\ x_4\ x_2\ x_2]$	$[x_4\ x_3\ x_2\ x_2]$	$[x_3\ x_2\ x_2\ x_2]$
$\lambda=\frac{\lambda_{max}}{30}$	σ	$[C_1\ C_4\ C_3\ C_2]$	$[C_1\ C_2\ C_3\ C_4]$	$[C_1\ C_2\ C_3\ C_4]$	$[C_1\ C_2\ C_3\ C_4]$
	$\mathbf{x}$	$[x_4\ x_4\ x_4\ x_4]$	$[x_4\ x_3\ x_2\ x_2]$	$[x_3\ x_2\ x_2\ x_2]$	$[x_2\ x_2\ x_2\ x_2]$
$\lambda=\frac{\lambda_{max}}{10}$	σ	$[C_1\ C_4\ C_3\ C_2]$	(C_1,x_1)	(C_1,x_1)	No
	$\mathbf{x}$	$[x_2\ x_2\ x_3\ x_4]$	No trans.	No trans.	No trans.

TABLE V
UTILITIES AND COMPUTATIONAL TIME ACHIEVED FOR DIFFERENT ORDERING ALGORITHMS

	Algorithm	Order obtained	Utility	Comp. time
Centralized	Optimal	$[C_6\ C_2\ C_1\ C_4\ C_3\ C_7\ C_5]$	100	>5min
	Heuristic	$[C_5\ C_6\ C_2\ C_7\ C_3\ C_1\ C_4]$	88	0.002s
	Greedy	$[C_4\ C_1\ C_6\ C_7\ C_2\ C_3\ C_5]$	95	0.002s
Decentralized	Safe Experimentation	$[C_6\ C_2\ C_1\ C_4\ C_3\ C_7\ C_5]$	100	2.09
	Partial Search	$[C_6\ C_2\ C_1\ C_4\ C_3\ C_7\ C_5]$	100	1.2s

for the last classifier, the impact of α to the total delay is less important and classifiers are ordered based on their accuracy to limit error propagation.

Finally, we can notice that as the misclassification c^F cost becomes more important, classifiers choose operating points with low probability of false alarm: as expected, classifiers choose their operating point where they will incur least penalty.

B. Comparison of the Performance and Processing Time Required by Different Algorithms

Heuristic Algorithm: It is intuitive to place classifiers with high accuracy and low processing time as early in the chain as possible. This limits error propagation, as well as reduces the average processing time per data item. One metric that captures the accuracy performance of a classifier is the Area Under Graph (AUG) of the DET curve. The AUG of a classifier goes from 1/2 for weak classifiers to 1 for perfect classifiers [4]. The delay performance of a classifier is captured by its processing complexity coefficient α. Based on these measures we design a heuristic algorithm where we order classifiers in decreasing order of AUG/α. We benchmark our results based on this heuristic algorithm.

Order Selected by Various Classifiers for Different Ordering Algorithms: The performance of the different ordering algorithms are shown in the Table V for 7 classifiers and fixed operating points per classifiers.[7]

As expected, centralized methods do not lead to the optimal order, yet, they require very little computational time. Heuristic method results in a 12% performance decrease. Decentralized algorithms converge to the optimal order, given that they ultimately browse through all the possible order, but in longer computational time. However, as will be shown in the next paragraph, convergence to a near-optimal order requires only a few iterations. Parametric Partial Search Algorithm (here with p =0, 1, $T=\infty$ and $\beta=0$ converges quicker than Safe Experimentation (here with ϵ =0, 1), to the optimal order. This will be discussed in more details in Section VIII-C.

System Compliance of Various Ordering and Operating Point Selection Algorithms: For ordering and operating point joint optimization, the optimal order cannot be achieved in a timely fashion. Indeed, for M queries, with N classifiers with A operating points each, $O(MN!A^N)$ 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 algorithm, shown on Table VI for 20 classifiers, each having 100 operating points.[8]

[7]Classifier's characteristics (p^F, p^D), ϕ, and α were generated randomly. $c^M=10$, $c^F=1$, $\lambda=0.1$. $t_0=g_0$ was selected to normalized optimal utility to 100.

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

TABLE VI
UTILITIES AND COMPUTATIONAL TIME ACHIEVED FOR DIFFERENT ORDERING ALGORITHMS

20 classifiers 100 operating points	SQP-Greedy algorithm	Safe Experimentation	Partial Search		
			$p = 0,01$	$p = 0,03$	$p = 0,05$
Time required for one iteration	0.18s	0.25s	0.34s	5.3s	37s

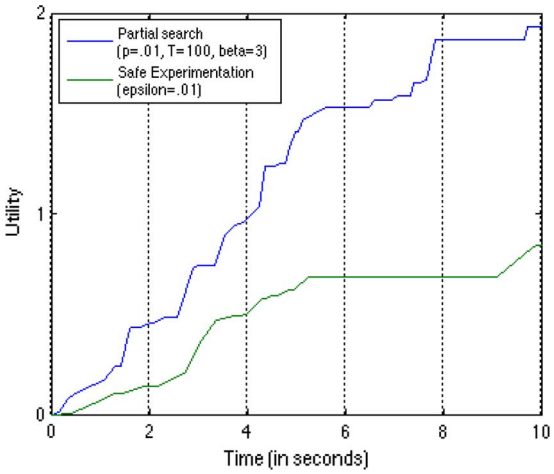


Fig. 5. Joint algorithms comparison.

In a multi-query context, SQP-Greedy needs to be performed for every query, while decentralized algorithms are performed by each classifier on-the-go by observation of their input data characteristics. Hence, to be able to compute orders in a fixed amount of time $\tau \leq \mathfrak{C}\tau^{\mathrm{dyn}}$, corresponding to the stream mining typical evolution characteristic time, the SQP-Greedy algorithm can only be performed for a fixed number of queries. For example, if the order must be reactualized every $\tau^{\mathrm{dyn}} = 1$ second, the SQP-Greedy algorithm performed for the system previously described can only process $\lfloor 1/.18 \rfloor = 5$ different queries. However, seven classifiers could process up to 2186 different queries. As a consequence, centralized algorithms are unadapted to multi-query context, since they can only process 0.25% of all possible queries. However, they can be used for systems requiring only one or a few queries. This limitation of SQP-Greedy in terms of number of queries simultaneously processable explains the need to vest in decentralized algorithms.

C. Decentralized Convergence Rate

Fig. 5 compares the utility achieved by the Safe Experimentation and the Partial Search algorithm for $N = 20$ and $A = 100$ actions.[9] To be able to adapt within a time of a second, Safe Experimentation was performed with an exploration rate $\epsilon = 0,01$ and Partial Search was performed with parameters $p = 0,01$, $T = 100$, and $\beta = 3$.

Both algorithms achieve increasing utilities over time. Yet, the Partial Search Algorithm achieves an approximately twice as quick convergence rate than the Safe Experiment Algorithm. This can be explained by the fact that Partial Search can probe multiple children within one iteration and can therefore learn more rapidly than Safe Experimentation, which only tries one order at each iteration.

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

IX. CONCLUSION

In this paper, we design adaptive algorithms to determine the optimal linear topology, and operating points, for a set of classifiers to trade off filtering accuracy and processing delay. Ordering the chain of classifiers through a proposed centralized Greedy algorithm leads to a close-to optimal utility but does not allow timely adaptation to system dynamics. In order to cope with these requirements, we proposed a decentralized approach where classifiers can make decisions autonomously with limited message exchanges. We examine several tradeoffs between the optimality of the solution and the convergence time. Specifically, we propose a Parametric Partial Search Algorithm, with control parameters on sampling the search space (all possible orders) combined with a reinforcement learning based technique that adapts exploration versus exploitation, i.e., much to search, how deep to search and whether to explore the tree or exploit known orders. We evaluate the performance of these algorithms in terms of determining the optimal order, as well as convergence time. There are several directions for future research, including extension of the approach to consider more generic topologies (e.g., trees, directed acyclic graphs, etc.), handling uncertainty in the information about classifier operating characteristics, data statistics, and resource variability. We are also in the process of implementing these algorithms on a large-scale distributed stream mining system [29].

APPENDIX

Result No 1: Recursive Goodput and Throughput Relationship: $\begin{bmatrix} t_k \\ g_k \end{bmatrix} = \underbrace{\begin{bmatrix} a_k & b_k \\ 0 & c_k \end{bmatrix}}_{T_k^{k-1}} \begin{bmatrix} t_{k-1} \\ g_{k-1} \end{bmatrix}$, with $a_k = p_k^F + (p_k^D - p_k^F)\overline{\phi_k}$, $b_k = (p_k^D - p_k^F)(\phi_k - \overline{\phi_k})$, and $c_k = p_k^D \phi_k$.

Proof: Introduce the notation $\mathcal{H}_{k-1}^{-} = \bigcap_{i=1}^{k-1} \mathcal{H}_i$.

The output $\hat{X}$ is a probabilistic function of input X. The proportion of correctly classified samples in $\mathcal{H}_k$ is captured by $p_k^D = \mathbb{P}(\hat{X} \in \mathcal{H}_k | X \in \mathcal{H}_k)$, while the proportion of falsely classified samples in $\mathcal{H}_k$ is $p_k^F = \mathbb{P}(\hat{X} \in \mathcal{H}_k | X \in \overline{\mathcal{H}}_k)$. Define $p^{++} = \mathbb{P}(X \in \mathcal{H}_k, X \in \mathcal{H}_{k-1}^{-} \text{ and } \hat{X} \in \mathcal{H}_k^{-},)$. Using Bayes'

formula

$$\begin{aligned}p^{++} &= \mathbb{P}\left((X,\hat{X}) \in \mathcal{H}_k | (X,\hat{X}) \in \mathcal{H}_{k-1}^-\right) \\ &\quad \times \mathbb{P}\left((X,\hat{X}) \in \mathcal{H}_{k-1}^-\right) \\ &= \mathbb{P}\left(\hat{X} \in \mathcal{H}_k | X \in \mathcal{H}_k \text{ and } (X,\hat{X}) \in \mathcal{H}_{k-1}^-\right) \\ &\quad \times \mathbb{P}\left(X \in \mathcal{H}_k | (X,\hat{X}) \in \mathcal{H}_{k-1}^-\right) \\ &\quad \times \mathbb{P}\left((X,\hat{X}) \in \mathcal{H}_{k-1}^-\right) \\ &= \underbrace{\mathbb{P}\left(\hat{X} \in \mathcal{H}_k | X \in \mathcal{H}_k\right)}_{p^D} \underbrace{\mathbb{P}\left(X \in \mathcal{H}_k | X \in \mathcal{H}_{k-1}^-\right)}_{\phi_k} \\ &\quad \times \underbrace{\mathbb{P}\left((X,\hat{X}) \in \mathcal{H}_{k-1}^-\right)}_{g_{k-1}}\end{aligned}.$$

We can similarly derive p^{+-}, p^{-+}, and p^{--}. Finally, we have $g_k = p^{++}$, and $t_k = p^{++} + p^{+-} + p^{-+} - p^{--}$. ■

Result No 2: Canonic Optimization Problem Formulation: The optimization problem (6) can be reformulated as

$$\max_{\sigma,\mathbf{x}} U(\sigma,\mathbf{x}) = g_N(\mathbf{x}) - K t_N^\sigma(\sigma,\mathbf{x}) - \sum_{k=1}^{N} \rho_{\sigma(k)} t_{k-1}^\sigma(\sigma,\mathbf{x})$$

subject to $0 \le \mathbf{x} \le 1$, where $K = (c^F/c^F + c^M) \in [0,1]$ and $\rho = (\lambda/c^F + c^M)\boldsymbol{\alpha} \in \mathbb{R}^{+N}$.

Proof: ϕ represents the probability of satisfying all classifiers and is therefore independent of the order, and the operating point selection. Developing $\begin{bmatrix} t_N^\sigma \\ g_N^\sigma \end{bmatrix} = T_N^\sigma T_{N-1}^\sigma \dots T_1^\sigma \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ leads to $g_N^\sigma = \prod_{k=1}^N (\phi_k^\sigma p_{\sigma(k)}^D) = (\prod_{k=1}^N \phi_k^\sigma)(\prod_{k=1}^N p_{\sigma(k)}^D) = \phi \prod_{k=1}^N p_k^D$, which is independent of the order. However, t_i^σ depends on the order and thus depends on σ and x_i^σ. Since r is proportional to t, it also depends on the order σ and operating points x_i^σ. We can develop the utility (6) $U_\lambda(\mathbf{x},\sigma)$ as

$$\begin{aligned}&- c^M[\phi - g_N(\mathbf{x})] - c^F[t_N^\sigma(\sigma,\mathbf{x}) - g_N(\mathbf{x})] \\ &\quad - \lambda \sum_{k=1}^{N} \left[\alpha_{\sigma(k)} t_{k-1}^\sigma(\sigma,\mathbf{x})\right] \\ &= (c^M + c^F) g_N(\mathbf{x}) - c^F t_N^\sigma(\mathbf{x},\sigma) \\ &\quad - \lambda \sum_{k=1}^{N} \alpha_{\sigma(k)} t_{k-1}^\sigma(\sigma,\mathbf{x}) - c^M \phi \\ &= (c^M + c^F) \left[g_N(\mathbf{x}) - \underbrace{\frac{c^F}{c^M + c^F}}_{K} t_N^\sigma(\mathbf{x},\sigma) \right. \\ &\qquad \left. - \sum_{k=1}^{N} \underbrace{\lambda \frac{\alpha_{\sigma(k)}}{c^M + c^F}}_{\rho_{\sigma(k)}} t_{k-1}^\sigma(\sigma,\mathbf{x}) \right] - c^M \phi\end{aligned}.$$

Hence, the optimization problem becomes: $(x^*,\sigma^*) = \arg\max_{x,\sigma}[g_N(x) - K t_N^\sigma(\sigma,x) - \sum_{k=1}^N \rho_{\sigma(k)} t_{k-1}^\sigma(\sigma, x_{k-1}^\sigma)]$. ■

Result No 3: Performance of the Greedy Algorithm: The value of the utility U_{ord}^G obtained with the Greedy Algorithm's order is at least 1/4th of the value of the optimal order

$$\frac{1}{\kappa} U_{\text{ord}}^{\text{opt}} \le U_{\text{ord}}^G \le U_{\text{ord}}^{\text{opt}} \quad \text{with } \kappa = 4.$$

Proof: We prove this by showing equivalence with a greedy 4-approximation algorithm for pipelined set-cover, as in [22].

After normalizing the coefficient of the objective function by t_0, the utility function (8) can be expressed as $U_{\text{ord}} = -(1/t_0)(\sum_{h=1}^N \rho_{\sigma(h)} t_{h-1}^\sigma + K t_N^\sigma) = -\sum_{h=0}^N \mu_h^\sigma (\prod_{j=1}^h \psi_{\sigma(j)})$. The problem defined in [22]

$$\underset{\sigma}{\text{minimize}} \sum_{i=1}^{N} t_i D_i \quad \text{where}$$

$$D_i = \begin{cases} 1 & i = 1 \\ \prod_{j=1}^{i-1} (1 - d(j|j-1)) & i > 1 \end{cases}$$

adapts to our setting by identifying $\mu_i \leftrightarrow -t_i$ and $\psi_i \leftrightarrow 1 - d(i|i-1)$. This finishes the proof. ■

Result No 4: Local Utility Expression: Each classifier $C_i = C_{\sigma(h)}$ will globally maximize the system's utility by autonomously maximizing its local utility $U_i = \underbrace{[v_h^\sigma \; w_h^\sigma]}_{=[v_i \; w_i]} \begin{bmatrix} t_{h-1}^\sigma \\ g_{h-1}^\sigma \end{bmatrix}$ where the local utility parameters $[v_h^\sigma \; w_h^\sigma]$ are defined recursively

$$\begin{aligned}[v_N^\sigma \quad w_N^\sigma] &= -\begin{bmatrix}\rho_{\sigma(N)} & 0\end{bmatrix} + [-K \quad 1] T_N^\sigma \\ [v_i^\sigma \quad w_i^\sigma] &= -[\rho_i \quad 0] + \begin{bmatrix} v_{i+1}^\sigma & w_{i+1}^\sigma \end{bmatrix} T_i^\sigma\end{aligned}.$$

Proof: By induction:

—

$$\begin{aligned}U_N &= -\rho_N t_{N-1} + g_N - K t_N \\ &= -[\rho_N \quad 0] \begin{bmatrix} t_{N-1} \\ g_{N-1} \end{bmatrix} + [-K \quad 1] \begin{bmatrix} t_N \\ g_N \end{bmatrix} \\ &= \underbrace{(-[\rho_N \quad 0] + [-K \quad 1] T_N(x_N^*))}_{[v_N \quad w_N]} \begin{bmatrix} t_{N-1} \\ g_{N-1} \end{bmatrix}.\end{aligned}$$

—

$$\begin{aligned}U_i &= -\rho_i t_{i-1} + U_{i+1} \\ &= -[\rho_i \quad 0] \begin{bmatrix} t_{i-1} \\ g_{i-1} \end{bmatrix} + [v_{i+1} \quad w_{i+1}] \begin{bmatrix} t_i \\ g_i \end{bmatrix} \\ &= \underbrace{(-[\rho_i \quad 0] + [v_{i+1} \quad w_{i+1}] T_i)}_{[v_i \quad w_i]} \begin{bmatrix} t_{i-1} \\ g_{i-1} \end{bmatrix}.\end{aligned}$$

■

Result No 5: Partial Search Algorithm Convergence: Under stable conditions the Partial Search Algorithm converges and the equilibrium point corresponds to a Nash equilibrium of the Markovian game. also, 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 $(\mathfrak{t})$. Under such assumption, we will keep track of the solution provided by the Partial Search Algorithm at each iteration: let $\mathbf{x}^{(\mathfrak{t})}$ and $\sigma^{(\mathfrak{t})} = \{C^{(\mathfrak{t})}_{0\rightarrow}, \ldots, C^{(\mathfrak{t})}_{N-1\rightarrow}\}$.

As a first step, we show that each iteration of Partial Search provides global utility improvement. We consider the $\mathfrak{t}+1$th iteration taking place. A subset of the overall trees has been selected, corresponding to probed children selection. the corresponding throughput and goodput values entering each node of the trees are computer.

We first consider the action of a classifier $C_{\sigma(N-1)}$ at penultimate position in this tree. This classifier has only one child C_j, which he will thus select as trusted child. Locally, $C_{\sigma(N-1)}$ determines the operating point ${x^*_{\sigma(N-1)}}^{(\mathfrak{t})}$ which will maximize its local utility by solving the optimization problem

$$ {x^*_{\sigma(N-1)}}^{(\mathfrak{t}+1)} = \arg\max_x U^{(\mathfrak{t}+1)}_{\sigma(N-1)}. $$

Doing so, it ensures that $U^{(\mathfrak{t}+1)}_{\sigma(N-1)} \geq U^{(\mathfrak{t})}_{\sigma(N-1)}$.

A classifier $C_{\sigma(N-2)}$ at $(N-2)$th position has two potential children C_{j_1} and C_{j_2}. It will probe each of them with probability $p^{\sigma(N-2)}_{j_1}$ and $p^{\sigma(N-2)}_{j_2}$ and accordingly compute the corresponding point $x^{(\mathfrak{t}+1)}_{\sigma(N-2),j_1}$ and $x^{(\mathfrak{t}+1)}_{\sigma(N-2),j_2}$, leading to utility $U^{(\mathfrak{t}+1)}_{\sigma(N-2),j_2}$ and $U^{(\mathfrak{t}+1)}_{\sigma(N-2),j_2}$. If one of the probed children brings classifier $C_{\sigma(N-2)}$ a local utility greater than its previously achieved utility, (for example, if both children are probed, if $\max(U^{(\mathfrak{t}+1)}_{\sigma(N-2),j_1}, U^{(\mathfrak{t}+1)}_{\sigma(N-2),j_2}) \geq \max(U^{(\mathfrak{t})}_{\sigma(N-1)})$, it will be the new trusted child at time $\mathfrak{t}+1$. By construction, we have $U^{(\mathfrak{t}+1)}_{\sigma(N-2)} \geq U^{(\mathfrak{t})}_{\sigma(N-2)}$.

Recursively, we derive similarly that $\forall h : U^{(\mathfrak{t}+1)}_{\sigma(h)} \geq U^{(\mathfrak{t})}_{\sigma(h)}$, and in particular, this is true for the source classifier. Given that $U = U_{\sigma(1)}$, this implies that $U^{(\mathfrak{t}+1)} \geq U^{(\mathfrak{t})}$, i.e., the Partial Search Algorithm is at global scale a best response algorithm and utilities are increasing over time. We insist on the fact that this is true only in the case of stable input characteristics, since the trusted child and operating point selection updates must always bring improved local utilities, which would not necessarily be the case in the case where stream characteristics vary. A consequence of this, the equilibrium point corresponds to a Nash equilibrium of the Markovian game, 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

$$ P(N) = \mathbb{P}\left(\{\text{For any } N \text{ classifiers, the Partial Search Ordering Algorithm converges}\}\right). $$

We want to prove by induction the proposition $P(N) = 1$.

Clearly, $P(1) = 1$, since there is only one order for one classifier. Suppose $P(N-1) = 1$, and consider a set of N classifiers. 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

$$\begin{aligned} P(N) &= \mathbb{P}\left(\exists \mathfrak{t} \text{ s.t. } \sigma^{(\mathfrak{t})} = \sigma_{\text{opt}}\right) \\ &= \mathbb{P}\left(\exists \mathfrak{t} \text{ s.t. } \sigma^{(\mathfrak{t})}(h) = \sigma_{\text{opt}}(h), \forall h \in [1,N]\right) \\ &= \mathbb{P}\left(\exists \mathfrak{t} \text{ s.t. } \sigma^{(\mathfrak{t})}(1) = \sigma_{\text{opt}}(1) | \sigma^{(\mathfrak{t})}(h) = \sigma_{\text{opt}}(h), \right. \\ &\qquad \left. \forall h \in [2,N]\right) \\ &\quad \mathbb{P}\left(\exists \mathfrak{t} \text{ s.t. } \sigma^{(\mathfrak{t})}(h) = \sigma_{\text{opt}}(h), \forall h \in [2,N]\right) \\ &\quad \mathbb{P}\left(\exists \mathfrak{t} \text{ s.t. } \sigma^{(\mathfrak{t})}(1) = \sigma_{\text{opt}}(1)\right) \\ &\quad \mathbb{P}\left(\exists \mathfrak{t} \text{ s.t. } \sigma^{(\mathfrak{t})}(h) = \sigma_{\text{opt}}(h), \forall h \in [2,N]\right). \end{aligned}$$

But $\mathbb{P}(\sigma^{(\mathfrak{t})}(2) = \sigma_{\text{opt}}(2), \ldots, \sigma^{(\mathfrak{t})}(N) = \sigma_{\text{opt}}(N)) = 1$ by induction. Furthermore,

$$\begin{aligned} &\mathbb{P}\left(\exists \mathfrak{t} \text{ s.t. } \sigma^{(\mathfrak{t})}(1) = \sigma_{\text{opt}}(1)\right) \\ &\quad = \lim_{\mathfrak{t}\rightarrow\infty} \sum_{t=1}^{\mathfrak{t}} p^0_{\sigma_o pt(1)} \left(1 - p^0_{\sigma_o pt(1)}\right)^t \\ &\lim_{\mathfrak{t}\rightarrow\infty} p^0_{\sigma_o pt(1)} \frac{1 - \left(1 - p^0_{\sigma_o pt(1)}\right)^{\mathfrak{t}+1}}{1 - \left(1 - p^0_{\sigma_o pt(1)}\right)} \\ &\lim_{\mathfrak{t}\rightarrow\infty} 1 - \left(1 - p^0_{\sigma_o pt(1)}\right)^{\mathfrak{t}+1}. \end{aligned}$$

Since $p^0_{\sigma_{\text{opt}}(1)} > 0$, $\mathbb{P}(\exists \mathfrak{t} \text{ s.t. } \sigma^{(\mathfrak{t})}(1) = \sigma_{\text{opt}}(1)) = 1$ and thus $P(N) = 1$. ∎

Result No 6: Depth Coefficient Determination:

$$ d^*_h = \begin{cases} 0, & \text{if } h < H \\ d_H, & \text{if } h = H \\ 1/p, & \text{if } h > H \end{cases} $$

where $H = N - 1 - \lfloor Np \rfloor$ and $d_H = N - \lfloor Np \rfloor / p$, and p is obtained by saturating the constraint $\tau^{it}(h) < \mathfrak{C}\tau^{\text{dyn}}$.

Proof: Since $\tau(p,d) = \tau^{ex} \sum_{h=0}^{N-1} ((N!/(N-h)!) p^h \prod_{g=1}^{h} d_g) = \tau^{ex}[1 + Npd_1(1 + (N-1)pd_2(1 + \ldots + (1 + pd_N)))]$, the low weights should be put on d_1, since it is a common factor in the expression of τ then d_2, etc, while the high weights should be put on the deeper classifiers. Hence, the first H classifiers should have a nul weight while the last $N - H - 1$ will be fixed maximal weight $1/p$. H is determined as the largest number such that $\sum d_h = N$, hence $Hp \leq N < (H+1)p$. Finally, d_H is the remaining weight $N - (1/p)(N - H - 1)$. ∎

Result No 7: Utility Bounds: $0 \leq U_\lambda(x^*, \sigma^*) \leq (1 - K - \sum_{i=1}^{N} \rho_i)\Phi$.

Proof: Choosing $\mathbf{x} = 0$ leads to $g_i = t_i = 0, \forall i \geq 1$ and therefore a utility of $-\rho_{\sigma(1)} t_0 \leq 0$. In this case, the data injector will prefer not to send any input stream ($t_0 = 0$). As a consequence, $U_\lambda(\mathbf{x}^*, \sigma^*) \geq U_\lambda(0, \sigma^*) = 0$. Given that $t_0 \geq t^\sigma_1 \geq \ldots \geq t^\sigma_N \geq g_N$

$$\begin{aligned} U_\lambda(\mathbf{x}^*, \sigma^*) &= g_N - K t^\sigma_N - \sum_{k=1}^{N} \rho_{\sigma(k)} t^\sigma_{k-1} \\ &\leq \left(1 - K - \sum \rho_k\right) g_N(\mathbf{x}^*) \end{aligned}$$

Then, $g_N(\mathbf{x}) = \prod_{k=1}^{N}(p_k^D \phi_k^\sigma) \leq \prod_{k=1}^{N}(\phi_k^\sigma) = \Phi$, which defines the upper bound for U. ■

REFERENCES

[1] L. Amini, H. Andrade, F. Eskesen, R. King, Y. Park, P. Selo, and C. Venkatramani, "The stream processing core," IBM T.J. Watson Research Center, Tech Rep. RSC 23798, Nov. 2005.

[2] B. Babcock, S. Babu, R. Motwani, and M. Datar, "Chain: Operator scheduling for memory minimization in data stream systems," in *Proc. ACM SIGMOD*, 2003.

[3] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press.

[4] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition*, 1997.

[5] S. Chaudhuri and K. Shim, "Optimization of queries with user-defined predicates," in *Proc. 22th Int. Conf. Very Large Data Bases*, 1996.

[6] Y. Chi, H. Wang, and P. S. Yu, "Loadstar: Load shedding in data stream mining," in *Proc. Int. Conf. Very Large Data Bases*, 2005.

[7] A. Condon, A. Deshpande, L. Hellerstein, and N. Wu, "Flow algorithm for two pipelined filter ordering problems," in *Proc. ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Syst.*, 2006.

[8] B. Foo and M. van der Schaar, *Distributed classifier chain optimization for real-time multimedia stream-mining systems*.

[9] F. Fu, D. S. Turaga, O. Verscheure, M. van der Schaar, and L. Amini, "Configuring competing classifier chains in distributed stream mining systems," *IEEE J. Sel. Topics Signal Process.*, vol. 1, no. 4, pp. 548–563, Dec. 2007.

[10] A. Gupta, K. Smith, and C. Shalley, "The interplay between exploration and exploitation," *Acad. Manage. J.*, 2006.

[11] J. M. Hellerstein and M. Stonebraker, "Predicate migration: Optimizing queries with expensive predicates," *ACM SIGMOD Rec.*, 1993.

[12] J. Hu and M. P. Wellman, "Multiagent reinforcement learning: Theoretical framework and an algorithm," in *Proc. 15th Int. Conf. Mach. Learn.*, 1998.

[13] J.-H. Hwang, S. Cha, U. Centimel, and S. Zdonik, "Borealis-r: A replication-transparent stream processing system for wide-area monitoring applications," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, 2008.

[14] H.-H. Lee, E.-W. Yun, and W.-S. Leeb, "Attribute-based evaluation of multiple continuous queries for filtering incoming tuples of a data stream," *Int. J. Inf. Sci.*, 2008.

[15] R. Lienhart, L. Liang, and A. Kuranov, "A detector tree of boosted classifiers for real-time objects detection and tracking," in *Proc. Int. Conf. Multimedia Expo (ICME)*, 2003.

[16] Z. Liu, S. Parthasarathy, A. Ranganathan, and H. Yang, "Near-optimal algorithms for shared filter evaluation in data stream systems," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008.

[17] Y. Mao, X. Zhou, D. Pi, Y. Sun, and S. T. C. Wong, "Multiclass cancer classification by using fuzzy support vector machine and binary decision tree with gene selection," *J. Biomed. Biotechnol.*, pp. 160–171, 2005.

[18] J. Marden, H. Young, G. Arslan, and J. Shamma, "Payoff based dynamics for multi-player weakly acyclic games," *SIAM J. Control Optim., Spec. Iss. Control Optim. Cooperative Netw.*, 2007.

[19] T. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[20] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly, "Detecting spam web pages through content analysis," in *Proc. 15th Int. Conf. World Wide Web*, May 2006.

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

[22] K. Munagala, I. Nishizawa, J. Widom, S. Babu, and R. Motwani, "Adaptive ordering of pipelined stream filters," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2004.

[23] L. Saul and M. I. Jordan, "Learning in Boltzman trees," *Neural Comput.*, 1994.

[24] T. E. Senator, "Multi-stage classification," in *Proc. 5th IEEE Int. Conf. Data Mining*, 2005, pp. 386–393.

[25] S. Seshadri, V. Kumar, B. Cooper, and L. Li, "A distributed stream query optimization framework through integrated planning and deployment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 10, pp. 1439–1453, Oct. 2009.

[26] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebarker, "Load shedding in a data stream manager," in *Proc. 29th Int. Conf. Very Large Data Bases*, 2003.

[27] D. Turaga, O. Verscheure, U. Chaudhari, and L. Amini, "Resource management for networked classifiers in distributed stream mining systems," in *Proc. IEEE ICDM*, 2006.

[28] D. S. Turaga, B. Foo, O. Verscheure, M. van der Schaar, and L. Amini, "Configuring competing classifier chains in distributed stream mining systems," *IBM Austin Center Adv. Studies*, 2008.

[29] D. S. Turaga, H. Park, R. Yan, and O. Verscheure, "Adaptive multimedia mining on distributed stream processing systems," in *Proc. IEEE Int. Conf. Data Mining*, 2009.

[30] V. V. Vazirani, *Approximation Algorithms*. New York: Springer-Verlag.

Raphael Ducasse received the M.S. degree in applied mathematics and electrical science from the Ecole Polytechnique, Paris, France, in 2008. He is currently pursuing the M.S. degree in electrical engineering at the University of California, Los Angeles.

Deepak S. Turaga received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Bombay, in 1997 and the M.S. and Ph.D. degrees in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 1999 and 2001, respectively.

He is currently a Research Staff Member in the Exploratory Stream Processing Department at the IBM T. J. Watson Research Center, Hawthorne, NY. He was with Philips Research during 2001–2003 and with Sony Electronics in 2003–2004. His research interests lie primarily in statistical signal processing, stream mining and machine learning, and multimedia coding and streaming applications. In these areas, he has published over 50 journal and conference papers and one book and two book chapters. He has also filed over 20 invention disclosures and has participated in MPEG standardization activities. He is also currently an Adjunct Assistant Professor in the Electrical Engineering Department, Columbia University, NY.

Dr. Turaga received the CSVT 2006 Transactions Best Paper Award (with M. van der Schaar and B. Pesquet-Popescu), and is a coauthor for the 2006 IEEE ICASSP Best Student Paper (with H. Tseng, O. Verscheure, and U. Chaudhari). He is an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, and an Associate Editor for the *Hindawi Journal for the Advances in Multimedia*. He was an Associate Editor of the IEEE TRANSACTIONS ON MULTIMEDIA during the period 2006–2008.

Mihaela van der Schaar is an Associate Professor in the Electrical Engineering Department, University of California, Los Angeles. Her research interests include multimedia communications, networking, and processing, distributed multimedia systems, multi-user communications and networking, as well as network economics and game theory.

Prof. van der Schaar received in 2004 the NSF Career Award, in 2005 the Best Paper Award from the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, in 2006 the Okawa Foundation Award, in 2005, 2007, and 2008 the IBM Faculty Award, the 2008 Exploratory Stream Analytics Innovation Award from IBM Watson, and in 2006 the Most Cited Paper Award from *EURASIP: Image Communications Journal*. She was an Associate Editor for the IEEE TRANSACTIONS ON MULTIMEDIA, IEEE TRANSACTIONS ON SIGNAL PROCESSING LETTERS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, *IEEE Signal Processing Magazine*, etc. She holds 32 granted U.S. patents and three ISO awards for her contributions to the MPEG video compression and streaming international standardization activities.