Querying Temporal Drifts at Multiple Granularities

Sofia Kleisarchaki, Sihem Amer-Yahia, Ahlame Douzal-Chouakria, Vassilis Christophides

Querying Temporal Drifts at Multiple Granularities

Sofia Kleisarchaki
LIG, Grenoble, France
Sofia.Kleisarchaki@imag.fr

Sihem Amer-Yahia
CNRS, LIG, Grenoble, France
Sihem.Amer-Yahia@imag.fr

Ahlame Douzal-Chouakria
LIG, Grenoble, France
Ahlame.Douzal@imag.fr

Vassilis Christophides
CSD, UoC, Greece & INRIA,
Paris-Rocquencourt, France
christop@csd.uoc.gr

ABSTRACT

There exists a large body of work on online drift detection with the goal of dynamically finding and maintaining changes in data streams. In this paper, we adopt a query-based approach to drift detection. Our approach relies on *a drift index*, a structure that captures drift at different time granularities and enables flexible *drift queries*. We formalize different drift queries that represent real-world scenarios and develop query evaluation algorithms that use different materializations of the drift index as well as strategies for online index maintenance. We describe a thorough study of the performance of our algorithms on real-world and synthetic datasets with varying change rates.

1. INTRODUCTION

Monitoring streaming content is a challenging big data analytics problem, given that very large datasets are rarely (if ever) stationary. In several real world monitoring applications (e.g., newsgroup discussions, network connections, etc.) we need to detect significant change points in the underlying data distribution (e.g., frequency of words, sessions, etc.) and track the evolution of those changes over time. These change points, depending on the research community, are referred to as *temporal evolution*, *non stationarity*, or *concept drift* and provide valuable insights on real world events (e.g. a discussion topic, an intrusion) to take a timely action. In this paper, we adopt *a query-based approach to drift detection* and address the question of processing *drift queries* over very large datasets. To the best of our knowledge, our work is the first to formalize flexible drift queries on streaming datasets with varying change rates.

In the problem of drift detection, given a number of m drifts ordered in time, we need no less than $m + 1$ intervals to detect them. Thus, without any assumption on the underlying distribution, we are interested in exploring how to segment the input stream in order to find a reasonable tradeoff between true positives and false negatives. Existing methods rely on segmenting the input stream, mostly into smaller fixed length intervals [4, 6, 11, 13, 18]. Although some works exist on partitioning the same stream into intervals of different granularities [2, 9, 19], they either adopt an offline analysis or they lack the ability of querying historical drifts in streams. A granularity in this case is an interval of time (e.g., every hour) or a number of observed data points (e.g., every 200 points). A drift is then defined as a significant difference in data distributions between two consecutive intervals at the same granularity. To detect drifts either statistical tests are directly applied on the data of two intervals [7, 13] or on their summaries, as for instance provided by a clustering algorithm [1, 4, 5]. To this end, two parameters impact the accuracy and efficiency of drift detection: the granularity of the intervals at which the original data items are clustered and the drift significance threshold used to assess whether or not there is a drift between two consecutive clusterings. In fact, fine-grained intervals can be used to capture the evolution of frequently changing streams. However, they may induce computation overhead for slowly changing ones. In addition, they may cause false positives, i.e., detecting drifts that are too abrupt and noisy, hence hurting precision. While a coarser granularity will improve precision, since more data is clustered in each interval, it may incur missing a drift that occurred at a finer granularity. Those misses will negatively affect recall. Moreover, the rate of change of a given dataset may vary over time thereby requiring to consider different clustering granularities and drift thresholds for the same dataset.

Understanding the tradeoff between precision (at higher segmentation granularities) and recall (at lower segmentation granularities), and the choice of thresholds to determine what constitutes a drift between two consecutive intervals of the same granularity, are the main objectives of this work. In this paper, we adopt an analytics approach in which we formalize drift queries over both fresh and historical data of arbitrary time granularities, in order to provide flexibility in tracking and analyzing drifts in evolving datasets. For this reason, we propose a flexible drift index to organize past data (or more precisely their summaries) at several granularities. Furthermore. we explore different creation strategies for this index relying on two common clustering approaches, namely independent [8, 15, 20] and cumulative [1, 5]. In independent clustering, data points belonging to a given interval are considered equally important and clustered independently. In cumulative clustering, data points in a given interval are

clustered with all previously occurring points and fresher data is more important than older data. Moreover, we propose different materialization strategies in order to explore the tradeoff between index storage and query response time.

Unlike existing approaches [4, 6, 11, 13, 18] comparing only the last most recent intervals, we exploit this index in order to identify drifts at different granularities. In particular, we formalize three kinds of queries: unary, refinement and synthesis aiming to detect drifts against historical data. A unary query is used to extract all drifts detected at a given granularity. A refinement query explores drifts from a source granularity (e.g., 5,000 points) to a finer target granularity (e.g., 500 points), iteratively. Such a query is useful to provide a more detailed description of drifts that have been detected in a high granularity, resulting in better recall. Synthesis queries, on the other hand, start from a relatively low granularity and summarize them into coarser ones. In this case, some of the particular details might be missed (low recall) in order to get drifts with higher precision. This flexibility in querying drifts addresses a long standing concern in detecting and tracking drifts in streaming content, that is, the ability to explore, in a declarative fashion, precision and recall tradeoffs at different granularities.

The evaluation of declarative drift queries relies on traversing the index of historical data summaries and, at each granularity, comparing its nodes pairwise to identify points where clusterings dissimilarity exceeds a threshold θ. Rather than setting drift thresholds a-priori [6, 17], we learn a θ-value for each dataset and at each granularity level in the index.

In summary, this paper makes the following contributions:

1. We introduce and formalize drift queries that provide high flexibility in analyzing precision and recall of drift detection for different time granularities.
2. We propose a drift index, a graph structure that captures change at different granularities and explore different materializations of the index that lead to the design of various index maintenance and query evaluation algorithms.
3. We propose learning algorithms for learning drift and clustering thresholds adaptively for different granularities and rates of change.
4. We perform a thorough study of proposed queries and indices using two real datasets, KDD Cup'99 [1] and Usenet [12], and a synthetically generated dataset. On the effectiveness front, our study confirms the need for our refinement and synthesis queries, as demonstrated by the very good precision/recall results they attain. On the scalability front, it validates the need for different materializations of the drift index in order to achieve a tradeoff between storage and query response time for datasets of varying change rates.

Section 2 defines our data model and queries. Section 3 describes the drift index, the online index maintenance algorithms and threshold learning. Section 4 is dedicated to query evaluation algorithms. Section 5 contains a description of our experiments and findings. The related work is summarized in Section 6. We conclude in Section 7.

2. DATA MODEL AND QUERIES

We are given a stream of data points $D = \{d_1, \ldots, d_i, \ldots\}$, $d_i = (tc_i, ts_i)$ where tc_i is an r-dimensional vector of attributes describing d_i and ts_i is the timestamp at which tc_i arrived. For example, on Usenet each attribute represents a term appearing in news feeds, while on KDD Cup'99 an attribute can be any feature (e.g., transmitted bytes, duration of connection) describing a connection record.

[1] *https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data*

2.1 Clustering and Drifts

Definition 1. **Time-Based and Point-Based Granularities.** A time-based granularity g is an interval of time. For example, g could be hourly, daily, bi-daily, or weekly. A point-based granularity g is an interval containing a fixed number of consecutive data points. For example, g could be 500 points or 1000 points. We assume a total order between granularities and use $g \prec\prec g'$ to denote that g' follows g. We also say that g' is coarser than g (and g is finer than g'). We write $g \prec g'$ to denote that g' immediately follows g when there does not exist a granularity between g and g'. In both cases, $g <> g'$.

Definition 2. **Time-Based and Point-Based Intervals.** Granularities are used to segment data points in D. A segmentation of a dataset D using a time-based or a point-based granularity g, results in a list of consecutives intervals denoted I_1^g, I_2^g, ... where I_i^g is the $i-$th interval (time-based or point-based) of granularity g.

For a daily granularity g applied to segment a week starting on Sunday, I_1^g corresponds to the time interval of Sunday. Respectively, I_1^g corresponds to the interval containing the first 500 data points when a point granularity $g = 500$ is used to segment incoming data.

The choice of point-based or time-based intervals to segment a dataset depends on the rate of arrival of data points. The main advantage of point-based intervals is the processing of data in fixed-size batches (in terms of number of points) although resulting intervals may have different lengths (in terms of time). Time-based intervals on the other hand, give the ability to tune the time granularity of the analysis (e.g., hour, day) resulting in fixed-length intervals (in terms of time) and varying in size (in terms of number of data points). Consequently, in order to generate intervals of comparable size, datasets that exhibit a high changing rate should be segmented with point-based intervals while more stable datasets can be segmented using time-based intervals.

Definition 3. **Granularity Clustering.** A granularity clustering $C^g(D)$ is a partitioning of all data points $d_i \in D$ into a set of clusterings $\{C_i^g, C_{i+1}^g, \ldots\}$ corresponding to consecutive, non-overlapping time intervals $\{I_i^g, I_{i+1}^g, \ldots\}$ at granularity g. A data point $d_i = (tc_i, ts_i)$ will belong to one interval I_j^g s.t. $ts_i \in I_j^g$. Here, tc_i is a vector of r-entries with the j-th entry corresponding to the weight of the j-th attribute (e.g., word). Each cluster $c \in C_i^g$ has a centroid, $center(c)$ which is itself an r-dimensional vector, where the j-th entry is the mean over the j-th entries of all data points in the cluster.

Definition 4. **Clustering Dissimilarity.** Given two clusterings C_i^g and C_j^g, we define, $d^2(c, c')$, the dissimilarity between a cluster $c \in C_i^g$ and a cluster $c' \in C_j^g$ as the Euclidean distance between their centroids:

$$\|center(c) - center(c')\|_2 \quad (1)$$

The dissimilarity between cluster $c \in C_i^g$ and a clustering C_j^g, $cdis(c, C_j^g)$, is defined as the closest cluster to c in C_j^g:

$$\arg\min_{c' \in C_j^g} d^2(c, c') \tag{2}$$

The dissimilarity between two clusterings, $dis(C_i^g, C_j^g)$, is defined as:

$$\frac{1}{|C_i^g|}\sum_{c \in C_i^g} cdis(c, C_j^g) + \frac{1}{|C_j^g|}\sum_{c' \in C_j^g} cdis(c', C_i^g) \tag{3}$$

where $|C_i^g|$ is the number of data points belonging to C_i^g.

Definition 5. **Drift.** For a dataset D, a granularity g, a threshold θ, we say that there is a drift between two consecutive intervals I_i^g and I_{i+1}^g, iff their associated clusterings C_i^g and C_{i+1}^g, satisfy $dis(C_i^g, C_{i+1}^g) \geq \theta$. We use $x_i^g = (I_i^g, I_{i+1}^g)$ to denote the pair of consecutive intervals for which there exists a drift and $X^g = \{x_1^g, x_2^g, \ldots\}$ for the set of all drifts detected at granularity g.

2.2 Drift Queries

The goal of drift queries is to compare drifts at different granularities and provide analysts with the ability to explore drift precison and recall across granularities. We study two kinds of queries, *refinement* and *synthesis*. Both kinds rely on a simpler *unary query* defined as follows.

Definition 6. **Unary Query.** A unary query $UQ(D, g)$ returns the set of all drifts X^g detected at granularity g for a dataset D.

Definition 7. **Refinement Query.** A refinement query $RQ(D, g_s, g_t)$ admits a source granularity g_s and a target one g_t s.t. $g_t \prec\prec g_s$, and returns a set of pairs $(x_i^{g_s}, x_j^g)$ where each drift $x_i^{g_s} \in X^{g_s}$ at g_s is associated to the finest corresponding drift $x_j^g \in X^g$ at a granularity g no finer than g_t as follows:

$\{x_j^g \in X^g, g_t \prec\prec g \prec\prec g_s \vee g = g_t \mid$
$\exists x_i^{g_s} \in X^{g_s}, I_j^g \subseteq (I_i^{g_s} \cup I_{i+1}^{g_s}),$
$\nexists x_k^{g'} \in X^{g'}, g \prec\prec g' \prec\prec g \vee g' = g_t, I_k^{g'} \subseteq (I_i^{g_s} \cup I_{i+1}^{g_s})\}$

where $I_i^{g_s} \cup I_{i+1}^{g_s} = [\min_{ts_j \in I_i^{g_s}}(ts_j), \max_{ts_k \in I_{i+1}^{g_s}}(ts_k)]$ and $I_j^g \subseteq (I_i^{g_s} \cup I_{i+1}^{g_s})$ if $\min_{ts_j \in (I_i^{g_s} \cup I_{i+1}^{g_s})}(ts_j) \leq \min_{ts_k \in I_i^{g_s}}(ts_k)$ and $\max_{ts_k \in I_i^{g_s}}(ts_k) \leq \max_{ts_j \in (I_i^{g_s} \cup I_{i+1}^{g_s})}(ts_j)$

Refinement queries provide a detailed analysis of drifts iteratively. For instance, for a source granularity $g_s = 1000$ connections on KDD Cup'99, selecting a granularity $g_t = 500$ might result in missing a more insightful analysis occurring at granularity $g_t = 100$. On the other hand, selecting $g_t = 100$ may result in retrieving false positives which could be avoided at $g_t = 500$. Therefore, the analyst will use the refinement query $RQ(D, 1000, 100)$ to obtain details of each drift at $g_s = 1000$ with a tradeoff between false negatives and false positives.

Definition 8. **Synthesis Query.** A synthesis query $SQ(D, g_s, g_t)$ admits a source granularity g_s and a target one g_t s.t. $g_s \prec\prec g_t$, and returns a set of pairs $(x_i^{g_s}, x_j^g)$ where each drift $x_i^{g_s} \in X^{g_s}$ at granularity g_s is associated to the coarsest corresponding drift $x_j^g \in X^g$ at a granularity g no coarser than g_t as follows:

$\{x_j^g \in X^g, g_s \prec\prec g \prec\prec g_t \vee g = g_s \mid$
$\exists x_i^{g_s} \in X^{g_s}, I_i^{g_s} \subseteq (I_j^g \cup I_{j+1}^g),$
$\nexists x_k^{g'} \in X^{g'}, g \prec\prec g' \prec\prec g_t \vee g' = g_t, I_i^{g_s} \subseteq (I_k^{g'} \cup I_{k+1}^{g'})\}$

Synthesis queries provide a summary analysis of drifts iteratively. For instance, for a source granularity $g_s = 100$ connections on KDD Cup'99, selecting a granularity $g_t = 1000$ might result in missing a more precise synthesis occurring at $g_t = 2000$. On the other hand, selecting $g_t = 2000$ can result in missing a summary of a drift, which could be obtained at $g_t = 1000$. Therefore, the analyst can use the synthesis query $SQ(D, 100, 2000)$ to obtain a summary of each drift at $g_s = 100$ with a tradeoff between false negatives and false positives.

3. DRIFT INDEX

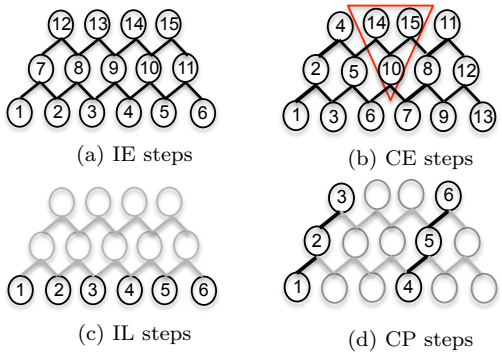
The flexibility of querying drift at different granularities requires the design of appropriate data structures able to capture clusterings at different granularities in such a way that queries are evaluated efficiently. In this section, we describe the *drift index*, an efficient graph data structure that is used to store and compute clusterings at different granularities. We first formalize the index and then study several materializations and develop algorithms for incremental index maintenance as data points continue to arrive.

Definition 9. **Drift Index.** The drift index is an undirected graph $G = (V, E)$ where each node contains a clustering C_i^g of points in D during interval I_i^g of granularity g. Given two different granularities g and g' s.t. $g \prec g'$, and two intervals I_i^g of granularity g and $I_j^{g'}$ of granularity g', there exists an edge in G between nodes C_i^g and $C_j^{g'}$ if $I_i^g \subseteq I_j^{g'}$.

Definition 9 does not necessarily impose an edge between nodes C_i^g and $C_j^{g'}$ every time $I_i^g \subseteq I_j^{g'}$ is satisfied. Indeed, different *materializations* of the index may be explored. The choice of which nodes to materialize affects three parameters: (i) the index size and hence the time it takes to build and maintain it as new data points arrive, (ii) the query response time, and (iii) the accuracy of query results. Since our approach is to serve queries for any time period, and not only the latest period at which data points arrived, the index continuously grows in size. Therefore, the key question we address in designing the index is: what are possible index materialization strategies, how much space they consume and how do they affect query evaluation (response time and accuracy)? In this section, we study index materialization alternatives. The impact of each index on query evaluation will be discussed in Section 4. In all our indices, the smallest granularity, g_{min}, is used to generate leaf-level nodes. In Section 5, we experiment with different values of g_{min}.

3.1 Full Index Materialization

When fully materialized, the drift index is a hierarchical structure where each level contains clusterings of data points inside intervals of the same granularity. Nodes corresponding to the finest granularity are leaves in the graph and each



(a) IE steps (b) CE steps

(c) IL steps (d) CP steps

Figure 1: Drift Index Variants

node, except nodes at the coarsest granularity, has one or two parents. For example, a node containing a clustering of data points for a 1-day granularity, e.g., Monday, will have two parent nodes each of which corresponds to a two-day granularity, in this case, Sun-Mon and Mon-Tue. Similarly, a node containing 1000 data points will have two parents, one containing it with the previous 1000 points and another containing it with the following 1000 points. More formally, given two granularities g and g' s.t. $g \prec g'$, and two intervals I_i^g and $I_j^{g'}$, there exists an edge from node C_i^g to node $C_j^{g'}$ iff $I_i^g \subseteq I_j^{g'}$.

Each node of the index contains a clustering of data points of a given granularity. Thus, an important aspect of index materialization is the selection of a clustering strategy and algorithm to generate the nodes. According to the clustering literature, timestamped datasets can be clustered in one of two ways. The first one, referred to as *the independent strategy*, encompasses a family of algorithms built upon the idea of visiting consecutive batches of data points by considering them as independent (e.g., one batch for Mon and another for Tue) and equally important in terms of arrival time (e.g., older data points are not penalized against fresher ones) [15, 8, 20]. The second approach, referred to as *the cumulative strategy*, parses data in a cumulative, single-pass fashion (e.g., data of Tue are clustered with those of Mon), aging older data points in a such a way that fresher data points are given more importance [1, 5]. Our exhaustive index is designed to work with any of the two clustering strategies which gives rise to two indices: Independent-Exhaustive (IE) and Cumulative-Exhaustive (CE).

3.1.1 Independent Exhaustive Index

IE is generated using the independent strategy where nodes of the same granularity, e.g., Sun, Mon, Tue, are produced using data points of consecutive, non-overlapping intervals. Nodes at the finest granularity level are produced by clustering the arriving data points, while nodes at coarser granularities are produced by summarizing the centroids of clusterings associated with lower granularities.

Figure 1a illustrates an instance of IE with nodes numbered in the order they are created. Algorithm 1 summarizes the different steps for building and maintaining the index. The algorithm takes as input the drift index G (empty at the beginning, non-empty in the case of index maintenance), a data stream D, a maximum granularity g_{max}, and an interval $I^{g_{min}}$ of minimum granularity g_{min}. For each batch of data point inside $I^{g_{min}}$, a clustering is produced (lines 3-6) using an independent clustering algorithm (e.g., DBScan [8] or k-means [10]). Nodes 1 to 6 of Figure 1a are produced by this step. Then, nodes at coarser granularities (e.g., nodes 7-15) are generated by applying the same algorithm over the centroids of clusters at lower granularity (lines 7-12).

Algorithm 1 IE Creation & Maintenance

Input: Drift index G, Stream of data points D, Max granularity g_{max}, Interval $I^{g_{min}}$ of min granularity g_{min}
Output: Updated drift index G'
1: $G' \leftarrow G$
2: $\{I_1^{g_{min}}, I_2^{g_{min}}, ...\} \leftarrow$ consecutive intervals at g_{min}
3: **for all** $I_i^{g_{min}}$ **do**
4: $C_i^{g_{min}} \leftarrow$ clustering in $I_i^{g_{min}}$ (e.g., DBScan, k-means)
5: Store $C_i^{g_{min}}$ in G' at granularity g_{min}
6: **end for**
7: **for all** $g_{min} \prec g \prec\prec g_{max}$ **do**
8: **for all** $C_i^g \in C^g$ **do**
9: $C \leftarrow$ clustering of the centroids of C_i^g and C_{i+1}^g
10: Store C in G' as the right parent of C_i^g
11: **end for**
12: **end for**
13: return G'

3.1.2 Cumulative Exhaustive Index

CE is generated using the cumulative strategy where nodes of the same granularity, e.g., Sun-Mon and Mon-Tue, are produced using data points from overlapping intervals. As a result, data points belonging to a given interval are clustered with previously occurring data points.

Figure 1b shows an instance of CE with its nodes numbered in the order they appear. Algorithm 2 summarizes steps of building and maintaining CE. The algorithm takes as input a drift index G (empty at the beginning, non-empty in the case of index maintenance), a data stream D, as well as the maximum granularity g_{max}, and the minimum granularity interval $I^{g_{min}}$ at g_{min}. After the initialization steps (lines 2-5), each data point is assigned to a cluster (line 7-8). Then every $I^{g_{min}}$ number of points (line 9), a corresponding clustering is generated and its centroid is stored in the index (line 11) at granularity g, which is incremented until g_{max}. Nodes 1, 2 and 4 of Figure 1b are generated by this step. When the maximum granularity g_{max} is reached a new path is initialized starting from the smallest granularity (line 16-21) and the process is repeated. Node 7 initializes this new path. In addition, for every node added in the index, its right sub-path to g_{min} is also produced (line 21). For instance, after the addition of node 4 in Figure 1b, its right sub-path consisting of nodes 5 and 6 is also added. These nodes are produced by applying the subtractive property [1] of clusters (i.e., subtracting clusters centroids). Finally, there is a set of nodes that do not belong to any right sub-path (e.g., nodes 10, 14, 15), forming the inverse triangle of Figure 1b. Each one of these nodes is generated after the addition of its right sibling and its sibling's sub-path. Lines 12-14 of Algorithm 2 illustrate this process using the additive property [1] of clusters (i.e., adding clusters centroids).

To handle infinite streams, several deletion strategies can be provided. A naive approach is to remove x intervals every X data points, including all corresponding nodes. However, this approach misses valuable historical data. For this reason, we consider an alternative deletion policy in which for

Algorithm 2 CE Creation & Maintenance

Input: Drift index G, Stream of data points D, Max granularity g_{max}, Interval $I^{g_{min}}$ of min granularity g_{min}
Output: Updated drift index G'

1: $G' \leftarrow G$
2: **if** G' is empty **then**
3: $C \leftarrow$ clustering of D (e.g., DBScan, k-means)
4: store C in G'
5: **end if**
6: $g \leftarrow g_{min}$
7: **for all** $d_i \in D$ **do**
8: $C \leftarrow$ clustering of d_i (e.g., CluStream)
9: **if** $(ts_i - ts_1)$ % $I^{g_{min}} == 0$ **then**
10: **if** $g \prec\prec g_{max}$ **then**
11: store C in G' at g-th granularity
12: **if** $\cup_{i=1,2..} I_i^g > g_{max}$ **then**
13: Build inverse triangle at g, by merging each node's left child with right-most child at g_{min}
14: **end if**
15: Move g to immediately following granularity
16: **else**
17: $C' \leftarrow C - C_j^{g_{max}}$ %Initializes path by subtracting the last stored $C_j^{g_{max}}$ from current clustering C
18: Store C' in G' at granularity g_{min}
19: Move g to granularity immediately following g_{min}
20: **end if**
21: Create right sub-path of node C
22: **end if**
23: **end for**
24: return G'

every X data points, the oldest x intervals are deleted and only their highest available node in the index is kept.

3.2 Partial Index Materialization

Since fully materialized versions of the drift index are expected to consume a lot of space, we propose Independent-Leaf (IL) and Cumulative-Path (CP) two partial index materializations, where fewer nodes are materialized thereby resulting in indices that are smaller in size.

The main idea in IL is to build nodes at the lowest granularity only (black nodes in Figure 1c), corresponding to lines 3-6 of Algorithm 1. All other nodes of higher granularities can be extracted from the leaf nodes at query time, if necessary. Respectively, the main idea in CP is to build paths containing all the nodes at higher granularities that include a given leaf node (black nodes in Figure 1d). The algorithm that builds and maintains CP is a modification of Algorithm 2, by ignoring lines 12-14 that build nodes of the inverse triangle and line 21 that builds the right sub-paths. From these nodes, built in partial materialization, all the remaining nodes (gray nodes in Figure 1d) may be generated at query time, if necessary (more details in Section 4).

3.3 Time & Space Complexity

The worst-case time complexity of IE and IL is dictated by DBScan, which needs $O(logn)$ time to find the neighbors for each of the n data points within an interval. Thus, the time complexity is $O(m * n * logn)$, where m is the number of nodes in the index. Furthermore, each cluster is represented by the statistics $(CF1; CF2; n)$ where $CF1$ and $CF2$ are r-dimensional vectors. Particularly, $CF1$ (resp, $CF2$) maintains, for each dimension, the sum of data values (resp, sum of the squares of data values). Thus, each cluster maintains $2r+1$ values and the space complexity is $O(K*(2r+1))$, where K is the number of clusters for all nodes.

The worst-case time complexity of CE and CP is specified by k-means, $O(n * k * d * i)$, where n is the number of r-dimensional data points forming k clusters at each interval and i the number of iterations. Furthermore, the space complexity is $O(m * k * (2 * r + 3))$, where $2r + 3$ values are maintained for each of the k-clusters of all m clustering nodes. These values contain the statistics described for IE and two extra values (details in [1]); the sum and the sum of the squares of the timestamps of input data.

Finally, the total number of nodes maintained in IE and CE is $m = \frac{1}{2} * L * (2 * |C^{gmin}| - L + 1)$, where $|C^{g_{min}}|$ is the number of clustering nodes at g_{min} and L is the number of index levels. The total number of nodes maintained in IL and CP is $m = \frac{N}{g_{min}}$, where N the total number of points.

3.4 θ and ϵ Learning

According to Definition 5, when the dissimilarity between two clusterings exceeds a threshold θ, a drift is detected. Since fixed threshold values are not always appropriate for data with varying drift rates, we are interested in learning θ experimentally and do so for each granularity of our index.

During the learning phase, a training dataset is used in order to estimate the drift parameter, θ. The training region is independent from the testing dataset over which queries are to be evaluated. Furthermore, the estimation of θ is automated and without any a-priori knowledge of the arrival rates of drifts. However, in order to be well-estimated, it should be learned on a long-enough time period to ensure capturing the occurrence of several drifts.

Algorithm 3 summarizes the learning process of θ_g values per granularity level g. The algorithm takes as input a drift index G, as well as minimum g_{min} and maximum g_{max} granularities for which θ_g values need to be estimated. For each granularity g within g_{min} and g_{max}, it extracts the distribution of dissimilarities X, based on Definition 4, between each pair of consecutive clusterings at g (line 3). Then, it performs DBScan (i.e., any other algorithm could be used, like k-means) over X, given as ϵ the average pairwise similarity of the 3-nearest neighbors in X. DBScan is performed 10 times, in order to select the clustering C that optimizes the $DunnIndex$ criterion. The Dunn index aims to identify dense and well-separated clusters. It is defined as the ratio between the minimal inter-cluster to maximal intra-cluster distance. The inter-cluster distance is defined as the average distance between the centroids of the clusters. Similarly, the intra-cluster distance is defined as the average distance of any pair of points inside each cluster. Finally, the value of θ_g is extracted by calculating the average Euclidean distances between clusters in C (line 10).

The precision of θ estimation could be challenged when the similarity between two consecutive clusterings (see line 3) varies significantly (i.e., bimodal distribution). This is due to the fact that the clustering distance is estimated (see line 10) by using the mean of clusters distribution and assuming a low and constant standard deviation over time. Applying a Z-Score statistical test over all training and testing intervals we observed that the variation of the majority of intervals (at least 95%) are less than two times the standard deviation from the mean for all granularities and real datasets.

Algorithm 3 Learning of θ-parameter

Input: Drift index G, Minimum granularity g_{min}, Maximum granularity g_{max}
Output: Parameter θ_g for each $g_{min} \prec\prec g \prec\prec g_{max}$

1: $minPts \leftarrow$ number of data dimensions
2: **for all** $g_{min} \prec\prec g \prec\prec g_{max}$ **do**
3: $X \leftarrow$ dissimilarities distribution between consecutive clusterings of g
4: $\epsilon \leftarrow$ avg pairwise similarity of 3 NN in X
5: **for all** $i \in [1, 10]$ **do**
6: $C \leftarrow DBScan(X, \epsilon, minPts)$
7: $dunnIndex \leftarrow DunnIndex(C)$
8: Pick C that maximizes $dunnIndex$
9: **end for**
10: $\theta_g \leftarrow$ average between clusters similarity of C
11: **end for**

We also propose a training phase to learn the ϵ parameter used by DBScan for building IE. Parameter ϵ defines a maximum ϵ-neighborhood for each cluster. In literature, a common way to choose its value is by plotting all distances to the nearest neighbors and selecting the value where the plot shows a strong bend. A similar approach is followed by our learning process, adapted to each granularity level. For brevity, we omit the algorithm for learning ϵ. The ϵ parameter can be estimated without any condition on the period's length. Thus, we propose to estimate both parameters (θ, ϵ) within the same wide-enough period.

It is worth noticing that the estimation of clustering parameters can quickly become outdated, particularly when dealing with rapidly evolving data distributions. In such cases, parameters re-estimation (e.g., every X intervals) may be useful to periodically adapt their values to data changes.

4. QUERY EVALUATION ALGORITHMS

This section presents our query evaluation algorithms using our proposed indices. Refinement and synthesis queries rely on unary queries that return a set of drifts X^g for any g. This is done by comparing the statistics between each pair of consecutive, non-overlapping clusterings at g using threshold θ_g. When a partial index is used (IL or CP), some index nodes (depicted in gray in Figures 1c, 1d) need to be generated on the fly possibly incurring computation overhead. The unary query algorithm is straightforward and is omitted for brevity. The performance of unary queries will be studied in detail in Section 5.

Algorithm 4 illustrates the steps for evaluating a refinement query. It takes as input any of the four materialized drift indices G and a range of granularities between g_s and g_t. Initially, it applies a unary query to detect all drifts $x_i^{g_s} \in X^{g_s}$ at g_s (line 2). Then, for each of these drifts, it detects all corresponding drifts $x_j^g \in X^g$ at finer granularities g, that are no finer than g_t (lines 3-11) using the condition $I_j^g \subseteq (I_i^{g_s} \cup I_{i+1}^{g_s})$ (line 8) for all drifts at g.

A synthesis query is evaluated over a drift index G and a granularity range between g_s and g_t, where $g_s \prec\prec g_t$. The steps of the algorithm are equivalent to Algorithm 4, by simply replacing the condition in line 8 with $I_i^{g_s} \subseteq (I_j^g \cup I_{j+1}^g)$. Thus, for any observed drift $x_i^{g_s}$ at g_s, a corresponding drift x_j^g at a coarser granularity g, no coarser than g_t is returned. The time interval $I_j^g \cup I_{j+1}^g$ of the corresponding drift x_j^g should take place during $I_i^{g_s}$ where $x_i^{g_s}$ was observed.

Algorithm 4 RefinementQuery

Input: Drift index G, Granularity range $[g_s, g_t]$ ($g_t \prec\prec g_s$)
Output: A set S of drift pairs $(x_i^{g_s}, x_j^g)$, $g \prec\prec g_s$

1: $S, prev, curr \leftarrow \emptyset$
2: $X^{g_s} \leftarrow UQ(D, g_s)$
3: **for all** $x_i^{g_s} \in X^{g_s}$ **do**
4: **for all** $g_s \prec\prec g \prec\prec g_t$ **do**
5: $prev \leftarrow curr$; $curr \leftarrow \emptyset$
6: $X^g \leftarrow UQ(D, g)$
7: **for all** $x_j^g \in X^g$ **do**
8: **if** $I_j^g \subseteq (I_i^{g_s} \cup I_{i+1}^{g_s}) \wedge I_{j+1}^g \subseteq (I_i^{g_s} \cup I_{i+1}^{g_s})$ **then**
9: $curr += x_j^g$
10: **end if**
11: **end for**
12: **if** $curr == \emptyset$ **then**
13: $S += (x_i^{g_s}, prev)$
14: break
15: **end if**
16: **if** $g == g_t$ **then**
17: $S += (x_i^{g_s}, curr)$
18: **end if**
19: **end for**
20: **end for**
21: return S

5. EXPERIMENTS

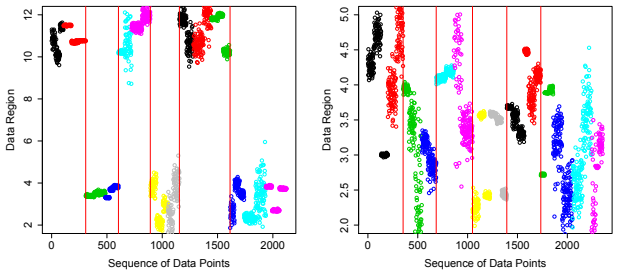
In this section, we provide a thorough investigation of our queries, both from the accuracy and the scalability perspectives. All experiments were conducted on a 2 GHz Intel Core i7 processor with 8 GB memory, which runs MAC operating system. Our accuracy results are the average of 5 consecutive runs. We learn ϵ and θ on a training dataset covering approximately 30% of the input data. Also, unless mentioned otherwise, we set the k parameter of CE to the average number of clusters produced by IE. Finally, we refer to each granularity level using incremental numbers (e.g., level 1 for the lowest granularity, then 2 etc). For both clustering algorithms (Clustream [1] and DBScan [8]), the implementations provided in MOA [3] are used. Some necessary extensions are applied in the implementation of CluStream, in order to provide additive and subtractive properties [1].

5.1 Dataset Preparation

5.1.1 Synthetic Datasets

We developed a synthetic data generator that provides the flexibility to produce datasets deriving from different distributions (i.e., well-separated, overlapping) and rates of change (i.e., sudden, incremental). Furthermore, the parameters of clustering are also tuned, including the number and size of clusters, as well as their density.

Specifically, synthetic datasets are produced with data points deriving from two distributions in a low and a high region. Each distribution consists of a number of close clusterings, that derive from the same region (i.e., low, high). Furthermore, each distribution contains a given number of data points and each clustering is described by k clusters. The total size of synthetic data is generated randomly and the number of drifts is also parameterized.



(a) Well-separated regions (b) Overlapping regions

Figure 2: Different Data Distributions

Three data sets are generated, in order to simulate different types of drifts. Figure 2 illustrates two examples. Specifically, Figure 2a depicts sudden drifts, marked with vertical lines, that occur each time the distribution of data oscillates between low and high region. The distribution of data in low region has values within [2, 4], while the distribution of high within [10, 12] forming well-separated regions. On the contrary, Figure 2b illustrates incremental drifts, occurring between low [2,4] and high [3, 5] regions of overlapping values. Finally, a third dataset is generated containing incremental drifts of consecutive regions, with values of low region within [2,4] and high within [4, 6].

For the purpose of building the index, the dataset is split into intervals of 200 data points each forming a leaf node at g_{min}. The next granularity consists of 400 points, the third of 600 and the 10th contains intervals of 2000 data points.

5.1.2 *Real Datasets*

Query accuracy was evaluated over two datasets derived from two different application domains. The first collection consists of 5,931 Usenet articles from the 20 Newsgroup collection where each article belongs to one of 6 news feeds (e.g., sports, science). A user can subscribe to any of these feeds, showing his interest in receiving relevant articles, or unsubscribe at any time. Each article is represented with a binary vector of 658 attributes, where each attribute indicates the absence or presence of a word. Another attribute indicates whether the user is interested in an article or not. Thus, the clustering procedure will result in clusters containing articles that are likely to derive from the same feed (e.g., sports) and interest the user. A drift is the moment where a user decides to unsubscribe from some feeds and subscribe to others and can be computed on the whole data. The ground truth hence is known and encompasses five drifts. Experiments on Usenet are performed using a small point-based interval of 100 data points, a maximum index depth of 6 and the number of clusters for CE is $k = 3$. The training set consists of 2,400 data points, containing 2 drifts.

The second dataset of KDD Cup'99 is a Network Intrusion detection stream of 494,020 normal TCP connections and cyber attacks. It contains a variety of intrusions that fall into 4 categories: DOS, R2L, U2R and PROBING. Most of the connections in the dataset are normal but occasionally bursts of attacks appear. Thus, we are interested in detecting drifts where realtime attacks occur. Each connection is described by 42 categorical (e.g., type of protocol) or continuous (e.g., bytes transmitted) attributes. For our analysis, we use the 34 continuous attributes. In order to create a ground truth for evaluating query accuracy, we consider as drifts the time moments where at least $minAttacks = 30$ consecutive malicious connections appear. The ground truth hence encompasses 45 drifts on the whole dataset. We set the smallest granularity to 500 points, the index depth to 10, corresponding to an interval length of 5,000 points and $k = 4$. The training dataset contains 20,500 points.

The smallest granularity is a critical parameter, indicated by the magnitude and arrival rate of drifts. To this end, the parameter settings used in [12] are applied for our experiments in Usenet, where 5 drifts exist within 5,931 points. On the contrary, wider intervals are selected for KDD Cup'99 of lower arrival rate with 45 drifts within 494,020 points.

5.2 Summary of Results

Our experiments show that unary queries can reach a 79% accuracy on real datasets. They also show that independent clustering attains a significantly better accuracy than cumulative for incremental changes of overlapping data distributions. They also confirm the usefulness of refinement and synthesis queries, by demonstrating their ability to explore the tradeoff between precision/recall. For instance, using CE, while $UQ(\text{KDD}, 1)$ and $UQ(\text{KDD}, 10)$ attain 52% and 13% accuracy respectively, $SQ(\text{KDD}, 1, 10)$ attains 74%. Moreover, the scalability evaluation of our indices show a tradeoff between full and partial materializations, in terms of index size and query response time. Fully materialized indices are at least an order of magnitude faster in query response time than partial. On the contrary, fully materialized indices require at least 4 times more space than partial.

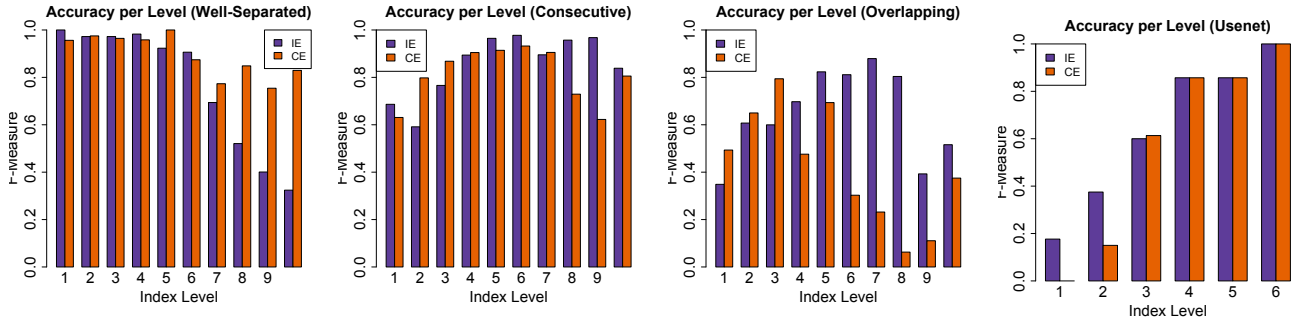
5.3 Accuracy of Drift Detection

Query accuracy varies between full and partial index materializations. This variation is caused by the random partitioning of data points during DBScan and k-means clustering. However, this variation is minimized with multiple executions and is not statistically significant. Hence we provide accuracy results for exhaustive indices only (IE, CE), assuming a not significantly different performance of partial indices (IL, CP). The accuracy is evaluated by using the traditional F-measure, which is the harmonic mean of precision and recall. A detected drift is considered a true positive if the corresponding real drift is within the compared intervals in the ground truth. This evaluation strategy is also used in [13]. For instance, a detected drift at point 800 extracted by comparing the point-based intervals [400, 800) and [800, 1200) will correspond to a real drift within the region [400, 1200). This drift can, for example, take place at point 1000. However, a drift at point 1000 might also be detected by comparing intervals [800, 1200) and [1200, 1600). Thus, in case of multiple detections of the same drift at a given granularity, we ignore its subsequent detections.

5.3.1 *Unary Queries*

Synthetic Data. The goal of synthetic data evaluation is to understand how different time granularities, as well as clustering strategies (independent, cumulative) affect query accuracy. To this end, we perform unary queries over different granularity levels and for both CE and IE.

Figure 3a illustrates the accuracy (y-axis) of unary queries for different granularities (x-axis) over the synthetic dataset with sudden drifts (Figure 2a). It shows that accuracy is very good for low granularities. However, recall worsens at



(a) Sudden drifts (b) Incremental (Consecutive) (c) Incremental (Overlapping) (d) Usenet dataset

Figure 3: Unary Queries Accuracy. A tradeoff between precision and recall is observed over different granularity levels.

higher granularities. Consequently, increasing the resolution of analysis decreases the ability of the algorithm to observe drifts occurring at finer granularities. However, precision remains greater than 0.9 at all levels.

Figure 3b depicts the unary queries behavior when applied on incremental drifts of consecutive data regions. We observe that the algorithm performs badly for very small or very large intervals. Very small intervals are sensitive to subtle changes causing a large number of false positives. Thus, those granularities suffer from low precision but exhibit high recall. Similarly, very wide intervals are susceptible to false negatives as they may miss drifts existing within them. On the contrary, intermediate granularities provide intervals that fit data better and improve accuracy.

The last dataset containing incremental drifts of overlapping regions (Figure 2b) reveals a statistically significant difference in accuracy (Figure 3c) between IE and CE for all levels greater than 2. The observed difference is due to the design of each index. For instance, CE tends to add input data into existing clusters. This addition causes cluster centroids to shift over time and absorb any change, considering it as non-significant. To alleviate that, we ran an experiment varying the number of clusters, k. Although not shown here, we observed no significant improvement in performance. Thus, the online and one-pass design of the algorithm causes the absorption of incremental changes. On the contrary, IE forms clusters by independently visiting data points in different intervals. The algorithm detects data regions of high density and is independent from previously computed clusters. Therefore, IE outperforms CE for overlapping data regions.

Although accuracy tends to decrease at higher index levels, there are some oscillations between levels. These fluctuations can be explained if we consider the statistical error introduced by adding and subtracting clusters' statistics as in [1]. A typical example of this error is illustrated in Figures 3a to 3c regarding the accuracy of CE at levels 9 and 10. Several nodes at level 9 are produced by the additive and subtractive property. On the contrary, none of the nodes at level 10 are generated by these properties. Thus, level 10 has a lower statistical error than level 9 and shows better accuracy despite its wider intervals.

Finally, we provide a comparison of our drift index accuracy with a state-of-the-art drift detection algorithm, named CUSUM [16]. CUSUM calculates the cumulative sum which detects a drift when the mean of the input data is significantly different from zero. Results show that our drift index outperforms CUSUM for each granularity and dataset. Specifically, CUSUM reaches a 98% of accuracy for the dataset of sudden drifts, while the accuracy drops in 24% and 16% for consecutive and overlapping datasets respectively.

Real Data. Figure 3d shows the F-Measure results of unary queries (y-axis) on Usenet for each level in the index (x-axis). The main trend observed is an increase in accuracy as the granularity increases. This is not surprising, as the frequency of drifts in this dataset occurs at least every 700 data points. Thus, when the interval length increases (especially to 600 points at g_{max}) the algorithm performs very well, as there are enough available points to detect the drift. In fact, the unary query at g_{max} reports three different drifts. A user who initially subscribed to electronics and crypt news changed her interests into hockey and sales and then subscribed to motorcycles and space news. It is worth mentioning that the inappropriateness of interval length at the leaf level induces CE to report no drifts.

Tables 1 and 2 illustrate the accuracy (last line) of unary queries for KDD Cup'99 for each granularity level of IE and CE respectively. Similarly to synthetic data, we observe that the finest and coarsest granularities cause a decrease in accuracy for both indices. Indeed, a low precision and a high recall characterize the leaf level and the inverse is observed at the coarsest granularity.

5.3.2 *Refinement & Synthesis Queries*

We designed refinement and synthesis queries to explore precision and recall tradeoffs. For brevity, in Tables 1 and 2 we present only the accuracy results for KDD Cup'99. Each row (resp. column) corresponds to g_s (resp. g_t) which are given as input to the queries. The tables contain every possible combination of levels in an attempt to discuss the impact of each query on accuracy. For example, a refinement query is evaluated over a small range of levels (e.g., $RQ(D, 2, 1)$), as well as on the entire index (e.g., $RQ(D, 10, 1)$). The upper-half of the tables contains the F-Measure of synthesis queries and the lower-half concerns refinement queries. The best accuracy results for each table are mentioned in circles, along with the corresponding unary queries results.

The question we attempt to answer is whether refinement and synthesis queries can attain a tradeoff between accuracy of unary queries at g_s or g_t. All values mentioned in bold indicate those cases. Thus, in the majority of the queries, the analyst will get a summary of drifts that exploits the tradeoff of precision and recall between g_s and g_t. However, note that there are few cases where the F-Measure is worse than any of the two levels. These cases are observed espe-

		g_t									
	Unary	1	2	3	4	5	6	7	8	9	10
1	0.50	-	**0.65**	**0.76**	**0.75**	**0.79**	**0.79**	**0.77**	**0.80**	**0.80**	**0.78**
2	0.62	0.49	-	**0.74**	**0.74**	**0.78**	**0.76**	**0.76**	**0.78**	**0.78**	**0.75**
3	0.73	0.48	0.60	-	**0.73**	**0.78**	**0.76**	**0.75**	**0.77**	**0.76**	**0.73**
4	0.73	0.49	0.61	**0.73**	-	**0.79**	**0.77**	**0.75**	**0.77**	**0.76**	**0.73**
5	0.79	**0.50**	**0.62**	**0.74**	**0.73**	-	**0.77**	**0.75**	**0.76**	**0.75**	**0.70**
6	0.77	0.49	**0.62**	**0.73**	**0.73**	**0.79**	-	**0.75**	**0.77**	**0.76**	**0.71**
7	0.75	**0.50**	**0.63**	**0.74**	**0.74**	**0.79**	**0.77**	-	**0.76**	**0.75**	**0.72**
8	0.76	**0.50**	**0.63**	**0.74**	**0.74**	**0.79**	**0.77**	**0.75**	-	**0.75**	**0.68**
9	0.75	**0.51**	**0.63**	**0.74**	**0.74**	**0.79**	**0.77**	**0.75**	**0.76**	-	**0.66**
10	0.44	0.36	0.41	**0.44**	**0.46**	**0.49**	**0.47**	**0.48**	**0.52**	**0.51**	-
Unary		0.50	0.62	0.73	0.73	0.79	0.77	0.75	0.76	0.75	0.44

g_s

Table 1: F-Measure for refinement (lower matrix) and synthesis (upper matrix) queries over **IE**, KDD Cup'99

		g_t									
	Unary	1	2	3	4	5	6	7	8	9	10
1	0.52	-	**0.63**	**0.72**	**0.73**	**0.69**	**0.68**	**0.68**	**0.68**	**0.71**	**0.74**
2	0.61	0.50	-	**0.73**	**0.72**	**0.70**	**0.71**	**0.70**	**0.74**	**0.75**	**0.78**
3	0.73	**0.52**	**0.61**	-	**0.73**	**0.71**	**0.73**	**0.74**	**0.72**	**0.73**	**0.77**
4	0.70	0.50	0.60	**0.70**	-	0.68	**0.68**	**0.68**	**0.72**	**0.72**	**0.75**
5	0.69	0.51	0.60	**0.71**	**0.72**	-	**0.66**	**0.68**	**0.66**	**0.68**	**0.76**
6	0.62	0.50	0.57	**0.68**	**0.67**	**0.63**	-	0.57	**0.63**	**0.58**	**0.65**
7	0.61	0.49	0.53	**0.65**	**0.62**	**0.64**	0.58	-	**0.56**	**0.58**	**0.67**
8	0.56	0.48	0.54	**0.65**	**0.64**	**0.62**	**0.56**	0.55	-	**0.48**	**0.56**
9	0.39	**0.44**	**0.48**	**0.55**	**0.49**	**0.48**	**0.45**	**0.43**	0.36	-	**0.37**
10	0.13	0.25	**0.20**	**0.25**	**0.19**	**0.22**	0.12	0.12	**0.16**	0.09	-
Unary		0.52	0.61	0.73	0.70	0.69	0.62	0.61	0.56	0.39	0.13

g_s

Table 2: F-Measure for refinement (lower matrix) and synthesis (upper matrix) queries over **CE**, KDD Cup'99

cially when refining queries at very low granularities (i.e., first two columns of tables). Low levels have a high rate of false positives, which negatively affects accuracy even when refining only the most precise drifts of higher levels.

5.4 Scalability Study of All Indices

We perform an index scalability experiment to study index size and build time and query response time. For this purpose, synthetic datasets of different sizes are produced.

5.4.1 Index Size

Index size depends exclusively on the number and size of maintained nodes. Therefore, increasing the number of index nodes, for instance by increasing the index depth or the volume of input data or by decreasing the interval length, will inevitably lead to an increase of the index size. Moreover, increasing the size of each node, for instance by increasing the number of clusters (k), will negatively impact index size. We fix the depth to 5 and we test the impact of the other parameters on the scalability of our indices.

Varying Dataset Size. Figure 4a illustrates the drift index size calculated in MBs (y-axis) as a function of input data size calculated in number of points (x-axis). We set the minimum interval size at g_{min} to 100 points for all datasets. The chart is in a log-log scale and we observe a linear trend of all indices as data size increases. We also observe that the lines of cumulative indices have a steeper slope. This is explained by the extra overhead paid for maintaining information about a cluster's origin. This information describes if a cluster is derived from others and is more likely to increase as the dataset increases. Finally, as expected, partially materialized indices consume at least 5 times less space.

Varying Interval Length. The interval length selected for each granularity has also a critical impact on index scalability. Figure 4b shows the result of varying the interval length within $[100, 1000]$ for a fixed dataset size of 225,000 points. Figure 4b reports a decreasing index size trend for all indices with wider intervals. When the size of an interval increases, the size of the clustering inside that interval is not affected. That is because the same clustering statistics are maintained. Thus, wider intervals reduce the number of nodes maintained by the index without affecting the size of each node. This explains the decreasing trend observed.

5.4.2 Time Results

The most frequent and time consuming operation of drift detection is the computation of clustering dissimilarities for every pair of nodes at each granularity. The cost of this operation naturally increases with the number of index nodes as shown in Figure 5a. Figure 5a illustrates the response time of all unary queries applied at each granularity for different dataset sizes. It is evident that IL and CP need at least an order of magnitude more time to detect drifts, as they produce missing nodes at query time.

Figure 5b illustrates the build time of each index, providing evidence for the trade-off between index building time and query response time. Figures 5b and 5a indeed show that the more time we spend building the index, the less time we need during query evaluation. Furthermore, Figure 5b illustrates that both independent indices, IE and IL, need more build time than cumulative ones, CE and CP. Despite the fact that IL is a partial index, it needs a higher build time than CE. This could be explained by the fact that IL iteratively visits data points in order to form clusters.

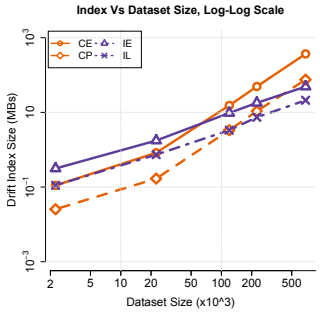
Figure 6a depicts unary query response time (y-axis) per granularity (x-axis) for a synthetic dataset of almost 225K data points. IL shows a sharp increase in drift detection as levels increase, due to generating missing nodes on the fly. On the contrary, IE's response time decreases at higher levels, due to fewer nodes. Thanks to the hierarchical structure of our indices less nodes exist at higher granularities, explaining the almost constant response time after level one for both indices. That also explains the sudden drop in response time for CP.

The performance of refinement queries is measured starting from level $g_s = 5$ until all finer levels of g_t, shown in the x-axis of Figure 6b. We notice that the longer the path from g_s to g_t, the more time it takes for the query to respond (for all indices). The same behavior is observed on synthesis queries, as shown in Figure 6c, where $g_s = 1$.

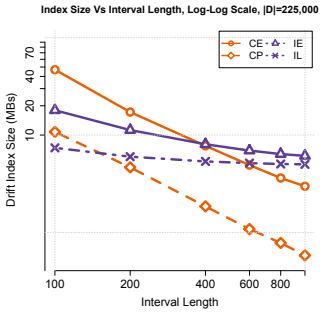
6. RELATED WORK

Most of the existing drift detection methods rely on segmenting the input stream into smaller fixed length intervals [4, 6, 11, 13, 18]. The comparison of intervals is then based either on statistical tests or on probabilistic measures. In the former case, a null hypothesis of equal distributions is formed [7, 13], while in the latter a user-defined threshold is utilized to detect drifts [6, 18]. However, a drawback of these works derive from the lack of dynamically adapting thresholds to the varying change rates. Furthermore, they suffer from the problem of single granularity, due to fixed length intervals, resulting in low precision/recall.

An attempt to solve the problem of single granularity is made in [14] by providing multi-partition techniques, but it also remains in an offline context. On the contrary, an online drift detection approach that aims to detect the most recent drifts, by comparing the last two intervals, is presented by FLORA2 [19]. FLORA2 dynamically learns interval length. A heuristic method shrinks the interval, by forgetting old data points, each time a drift occurs; otherwise, the interval

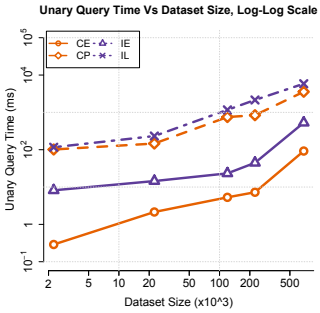


(a) Index size Vs dataset size

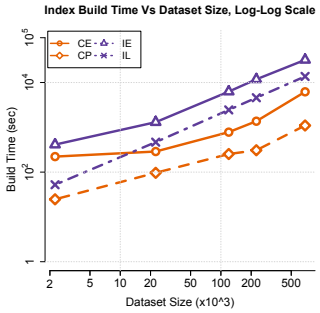


(b) Index size Vs interval length

Figure 4: Index Size

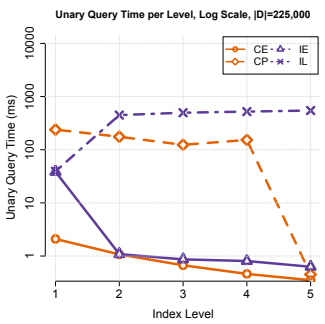


(a) Unary time Vs dataset size

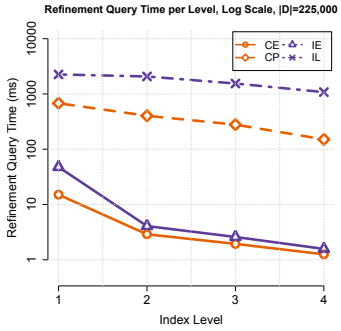


(b) Index build time

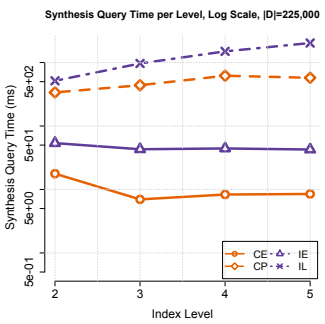
Figure 5: Unary Query Response and Index Build Time



(a) Unary query time response



(b) Refinement query time response



(c) Synthesis query time response

Figure 6: Query Time. Less nodes at higher levels of hierarchical index reduce the corresponding query time response.

grows. Also, some other approaches exist [2, 9] calculating statistics over sliding and growing intervals in order to detect a drift. However, all these works lack the flexibility of querying historical and fresh data for detecting drifts at different granularities.

7. CONCLUSION

We present drift queries on streaming content at different time granularities. This flexibility in querying drifts addresses a long standing concern, that is, the ability to explore, in a declarative fashion, precision and recall tradeoffs introduced by data segmentation at different time granularities. Our drift index enables efficient query evaluation and our experiments on real and synthetic datasets show the usefulness of our queries as demonstrated by the very good precision/recall results they attain.

We believe that this work laid the foundation for a series of new contributions in querying drifts in streaming content. One direction we are pursuing is the applicability of our approach to the detection of sales drifts in the retail industry and the ability to compare drifts over time across multiple products and stores.

8. REFERENCES

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proc. of the 29th international conference on VLDB - Volume 29*, pages 81–92, 2003.

[2] A. Bifet. Adaptive stream mining: Pattern learning and mining from evolving data streams. In *Proc. of the 2010 Conference on Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*, pages 1–212, Amsterdam, The Netherlands, 2010. IOS Press.

[3] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. Moa: Massive online analysis. *J. Mach. Learn. Res.*, 11:1601–1604, Aug. 2010.

[4] A. Bondu, B. Grossin, and M.-L. Picard. Density estimation on data streams : an application to change detection. In *EGC 2010*, pages 229–240.

[5] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *2006 SIAM Conference on Data Mining*, pages 328–339, 2006.

[6] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi. An information-theoretic approach to detecting changes in multi-dimensional data streams. In *Proc. Symp. on the Interface of Statistics, Computing Science, and Applications*, 2006.

[7] A. Dries and U. RÃijckert. Adaptive concept drift detection. *Statistical Analysis and Data Mining*, 2(5-6):311–327, 2009.

[8] M. Ester, H.-P. Kriegel, J. S, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[9] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In A. Bazzan and S. Labidi, editors, *Advances in Artificial Intelligence – SBIA 2004*, volume 3171 of *Lecture Notes in Computer Science*, pages 286–295. Springer Berlin Heidelberg, 2004.

[10] J. A. Hartigan and M. A. Wong. A K-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.

[11] N. Japkowicz, C. Myers, and M. Gluck. A novelty detection approach to classification. In *Proc. of the Fourteenth Joint Conference on Artificial Intelligence*, pages 518–523, 1995.

[12] I. Katakis, G. Tsoumakas, and I. Vlahavas. Tracking recurring contexts using ensemble classifiers: an application to email filtering. *Knowledge and Information Systems*, 22(3):371–391, 2010.

[13] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *Proc. of the 13th International Conference on VLDB - Volume 30*, pages 180–191, 2004.

[14] M. M. Masud, L. Khan, and B. Thuraisingham. A multi-partition multi-chunk ensemble technique to classify concept-drifting data streams. In *Advances in Knowledge Discovery and Data Mining*. Springer, 2009.

[15] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *Data Engineering, 2002. Proc. 18th International Conference on*, pages 685–694, 2002.

[16] E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):pp. 100–115, 1954.

[17] R. Sebastião and J. a. Gama. Change detection in learning histograms from data streams. In *Proc. of the Aritficial Intelligence 13th Portuguese Conference on Progress in Artificial Intelligence*, EPIA'07, pages 112–123, Berlin, Heidelberg, 2007. Springer-Verlag.

[18] P. Vorburger and A. Bernstein. Entropy-based concept shift detection. In *ICDM*, pages 1113–1118. IEEE Computer Society, 2006.

[19] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. In *Machine Learning*, pages 69–101, 1996.

[20] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. *SIGMOD Rec.*, 25:103–114, June 1996.