Quality-Driven Resource-Adaptive Data Stream Mining*

Conny Junghans
Institute of Computer Science
Heidelberg University,
Germany
cj@uni-hd.de

Marcel Karnstedt
DERI - Digital Enterprise
Research Institute
National University of Ireland,
Galway
marcel.karnstedt@deri.org

Michael Gertz
Institute of Computer Science
Heidelberg University,
Germany
gertz@uni-hd.de

ABSTRACT

Data streams have become ubiquitous in recent years and are handled on a variety of platforms, ranging from dedicated high-end servers to battery-powered mobile sensors. Data stream processing is therefore required to work under virtually any dynamic resource constraints. Few approaches exist for stream mining algorithms that are capable to adapt to given constraints, and none of them reflects from the resource adaptation to the resulting output quality. In this paper, we propose a general model to achieve resource *and* quality awareness for stream mining algorithms in dynamic setups. The general applicability is granted by classifying influencing parameters and quality measures as components of a multiobjective optimization problem. By the use of CluStream as an example algorithm, we demonstrate the practicability of the proposed model.

1. INTRODUCTION

A crucial challenge in data stream mining is to cope with the effects of dynamics [1; 12; 18; 21]. Data stream management systems (DSMS) [2; 16] run several continuous queries, each containing various mining operators. With a changing set of queries, the amount of resources assigned to each of them has to be adjusted. Consequently, the amount of resources for each operator contained in the queries should be adapted, which is the approach that we assume in this work. Stream mining algorithms running on a dedicated machine have to cope with variable stream rates. The usually fixed available resources should be utilized optimally to achieve mining results of highest possible quality in each situation. In wireless sensor networks, where sensors usually have some memory and processing capabilities, local resources are also fixed. Additional constraints are posed by limited bandwidth and battery lifetime of wireless devices.

Attempts to deal with this challenge led to the development of so called *resource-adaptive*, or resource-aware, stream-mining algorithms. Such algorithms are aware of the available resources and the dynamically changing variables, such as the stream rate and the patterns discovered in the data stream. Most work in this area has focused solely on minimizing resource utilization. A major problem is that the effect of these techniques on the mining quality is often ignored, i.e., no user-defined quality constraints are considered and the quality of the mining results is often unknown. Thus, they lack any indication how the adaptation affects accuracy and reliability of the mining results. Further, most existing approaches focus on decreasing the algorithm's resource requirements. With a look at the mining quality, it should be possible to also increase resource utilization when sufficient resources exist.

Most stream-mining algorithms follow a three layer approach, as illustrated by the shaded part in Figure 1. The *online mining* component analyzes the incoming data stream, which might be a filtered substream of the raw data stream (obtained by, e.g., sampling or load shedding [6]). The results of the online mining component are stored in a summary data structure provided by the second layer, called *synopsis*. Examples of synopses are sketches, windows, and dedicated data structures like the pattern tree used in FP-Stream [13] and the snapshot pyramid used in CluStream [1]. Finally, the *offline mining* component answers user queries by accessing information stored in the synopsis. Thus, the offline mining component is usually not constrained by the one pass requirement as the online component is.

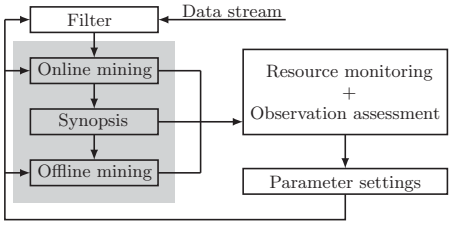

Figure 1: Extended three-layer model

So far, most of the proposals for resource-adaptive stream mining suggest a specialized solution for particular algorithms, i.e., adding resource awareness to the online or synopsis component in Figure 1. The field lacks a framework for achieving resource adaptivity for general stream-mining algorithms. In this work, we suggest such a framework, which considers quality awareness as a crucial requirement. The general principle is to identify and appropriately adapt parameters of stream-mining algorithms that influence resource requirements and mining quality. This results in an extension to the three-layer model as shown on the right of Figure 1. The *resource monitoring and observation assessment* component collects information about the current system state. Based on this, it is decided whether parameters have to be adapted (cf. Section 7). The actual pa-

*This work was partly funded by the Science Foundation Ireland under grant no. SFI/08/CE/I1380 (LION-2).

rameter adaptation takes place in the *parameter adaptation* component (cf. Sections 5 and 6). The new parameters are set in the stream-mining algorithm and the stream analysis continues until the adaptivity component is activated again. Of course, in addition to the benefits of resource-adaptive stream mining, automated parameter adaptation also entails additional resource requirements. We believe that this additional overhead is a very low price for the gained flexibility. In general, we assume that the additional resources are available. Alternatively, special resources might be reserved for this subtask, but an appropriate estimation might pose a problem again. Especially in situations when the workload of the stream-mining algorithm is high, the overhead imposed by triggering the resource adaptivity component may sum up to a CPU cycle requirement that exceeds the constraints. Throughout this work, we will point out specific issues in this context and briefly discuss methods to overcome them.

The extended model is resource-aware and quality-driven, meaning that we aim at maximizing the mining quality while observing the given resource limits. To this end, the proposed model is based on a multiobjective optimization problem with two conflicting objectives: (1) maximizing mining quality, and (2) minimizing resource utilization. The result of the optimization problem is a set of parameter values. We present a way to solve this optimization problem at certain points in time during the stream mining process and exemplarily demonstrate how to apply our model to the popular stream mining algorithm CluStream [1].

The proposed model is designed to extend stream mining algorithms that are not yet resource-adaptive as well as to cover existing approaches for resource awareness. The concrete contributions of the presented work are:

1. We formalize the data stream mining process, all variables involved, and the aspects that influence an algorithm's resource requirements and mining quality.
2. We provide a comprehensive and generally applicable model that captures all adaptation factors and their interactions. The model is designed to determine parameter values that lead to best mining quality with respect to given resource constraints.
3. We propose to solve the problem of finding parameter values as a multiobjective optimization problem and suggest one concrete approach for solving it.

The remainder of this paper is structured as follows. In Section 2 we briefly present the CluStream algorithm, which we use as a running example. Section 3 describes related work on resource-aware stream mining. Section 4 summarizes commonalities of stream-mining algorithms and discusses resource requirements and quality measures. In Section 5, we present our model for resource-adaptive stream mining and show that the optimal parameter settings can be determined by solving a multiobjective optimization problem. We present an approach to solve such an optimization problem in Section 6. Section 7 discusses two different adaptivity schedules, answering the question *when* parameter adaptation should be conducted. Finally, we conclude in Section 8 and highlight open issues.

2. RUNNING EXAMPLE

To demonstrate the application of our model, we use the clustering algorithm CluStream [1] as a running example, as it has been used previously to demonstrate resource-adaptive extensions [10; 12]. CluStream is a well-known and popular algorithm that we expect many readers to be familiar with. In the following we present a brief summary of the original algorithm. In previous work [9; 10], we also present resource-adaptive extensions to algorithms for frequent itemset mining and outlier detection in data streams, using the model presented here.

The design of CluStream follows the 3-layer model introduced in Section 1. In the online mining component, CluStream maintains q micro-clusters, which are updated as new stream objects arrive. q is an input parameter and should be significantly larger than the number of natural clusters in the data, but also much smaller than the number of data points arriving in the data stream [1]. During the initialization phase, the initial q clusters are created using a traditional clustering algorithm for static data sets, e.g., k-means, on the first points of the stream. After that, existing clusters are deleted or merged whenever a new cluster needs to be added.

At certain points in time, snapshots of the current micro-clusters are stored in a synopsis called *pyramidal time frame* (PTF). In the offline mining component, users query the PTF by requesting k clusters, $k < q$, in the past time horizon h before current time t_c. A k-means clustering algorithm is used to determine these clusters based on the available snapshots. The PTF uses a tilted time window model. The i-th level of the PTF contains the $\alpha^l + 1$ most recent snapshots taken at clock times that are divisible by α^i, $\alpha \in \mathbb{N}$, $\alpha \geq 1$. As an example, if $\alpha = 2$ and $l = 2$, then each level of the PTF contains at most $2^2 + 1 = 5$ snapshots.

The accuracy of the clustering depends on the order, i.e., the level in the PTF, of snapshots available for the queried time interval. The beginning of the queried time interval, i.e., $t_c - h$, has to be approximated by the last stored snapshot t_s of any order before time $t_c - h$. The accuracy of this approximation is bound by the following inequality:

$$t_c - t_s \leq (1 + 1/\alpha^{l-1}) \cdot h$$

3. RELATED WORK

Resource adaptivity has been proposed in the context of individual algorithms, e.g., in [10; 17; 18; 20], and as frameworks that facilitate resource adaptivity in arbitrary stream-mining algorithms [10; 12; 15] and in DSMSs [3; 4].

Individual resource-adaptive stream-mining algorithms have been proposed by, e.g., Teng et al. [18] for frequent temporal patterns, Lee and Lee [17] for frequent itemsets, Vlachos et al. [20] for periodicity estimation, and Franke et al. [10] for frequent itemsets and clustering. In these algorithms, the techniques used to facilitate resource adaptivity are specific to the proposed algorithm and cannot be directly applied to other algorithms, i.e., there is no underlying generic model for resource adaptivity. In addition, all but the algorithm in [10] lack quality-awareness and therefore no estimation of the result quality is provided. The emphasis is usually on handling situations where resources are scarce, e.g., in [17; 20], and often only one resource, usually memory or CPU-time, is considered. Only in [10] attention has been paid to facilitate the utilization of excess resources.

Resource-adaptive operator scheduling in DSMSs has been proposed by, e.g., Babcock et al. [3] and Berthold et al. [4]. In [3] a load-aware operator scheduling strategy is proposed.

Using estimates for the selectivity and per-tuple processing time of each operator, operators are scheduled such that the amount of tuples that are present in the DSMS is decreased as fast as possible. The approach in [4] has similar objectives as our research and focuses on meeting Quality of Service (QoS) requirements for a given query plan. The authors present a model that allows to calculate the resource requirements of the operators in a fixed query plan and subsequently parameterize the query plan such that QoS requirements are met. However, the paper only considers simple operators like joins, aggregation, and grouping, and does not include stream mining techniques.

General frameworks for resource-adaptive stream mining have been proposed by Jain et al. [14], Karnstedt et al. [15], and Gaber and Yu [12]. In [14] it is proposed to perceive resource management in stream processing as a filtering problem, and use Kalman Filters to reduce the amount of data that need to be processed. In previous work [15], Karnstedt et al. proposed a preliminary version of the model for quality-driven resource adaptive stream mining presented in this paper. This model formalizes the techniques used in [10] to create a generic framework that can be applied to a variety of stream mining algorithms. However, the model proposed in [15] does not yet consider parameter adaptation as a multiobjective optimization problem. In [12] a model is proposed that uses algorithm granularity (AG) settings to adapt an algorithm's resource consumption to the amount of available resources. The AG model focuses on situations where resources are scarce. AG has been classified into three classes, which we will detail in Section 4.1, where we reuse the notion of these three classes. The AG model is not quality-aware. Also, AG does not take the correlations between variables in stream mining into account. The model proposed in this paper therefore subsumes the model proposed in [12]. The AG model in [12] was used to develop a resource-adaptive clustering algorithm, RA-Cluster, which is an extension of the CluStream algorithm. RA-Cluster uses randomization to decrease the CPU-time demand. In contrast, our approach aims at adapting the existing variables, thus influencing all resource requirements and enabling us to not only alleviate critical situations where resources are scarce, but also to utilize excess resources.

4. CHARACTERISTICS

In the following, we summarize characteristics of data stream-mining algorithms. First, we discuss aspects that influence the resource requirements of stream-mining algorithms in Section 4.1. Then, we introduce quality measures that are used to assess the quality of stream mining in Section 4.2.

4.1 Resource Requirements

In a data stream mining application several resources are constrained and thus have to be carefully allocated. The main resources are:

Main memory The synopsis and all intermediate results must fit into main memory.

CPU cycles Stream processing must keep up with the pace of the stream, otherwise uncontrolled data loss occurs because stream elements cannot be buffered indefinitely. Query answering requires CPU cycles as well.

Bandwidth Bandwidth is limited in wireless sensor networks, but also has to be considered in wired networks when large amounts of data need to be transferred.

Battery power CPU utilization, memory access, and data transfer consume battery power. Hence, these consumers should be used prudently on mobile devices like sensors.

We denote the set of all resources by $\mathcal{R}$. If $R \in \mathcal{R}$ is a constrained resource, there is an upper limit $R^{\top}$ associated with this resource. Similarly, a lower limit $R^{\perp}$ may be associated with each resource $R \in \mathcal{R}$. The sets of all upper and lower limits on resources in $\mathcal{R}$ are denoted $\mathcal{R}^{\top}$ and $\mathcal{R}^{\perp}$ respectively. Accordingly, we use the notation $\mathcal{R}^{\mp}$ to refer to all given constraints, i.e., both, upper and lower limits, of $\mathcal{R}$. While a lower limit $R^{\perp}$ might appear uncommon, we include it in our model to make it as flexible and general as possible. For example, the frequent itemset mining algorithm proposed in [17] uses a lower bound on the desired memory usage. In general, it is not required that both constraints $R^{\top}$ and $R^{\perp}$ are defined for each $R \in \mathcal{R}$.

Three aspects $A1$, $A2$, and $A3$, which roughly correspond to the three basic layers introduced in Figure 1, influence the resource requirements of any data stream-mining algorithm. They represent classes of "tuning knobs" for the resource requirements of a stream-mining algorithm.

A1: Stream properties. Properties of the stream include the stream rate and characteristics of its individual elements, such as value range, distribution, and size. The effective rate of a stream to process can be changed using methods like sampling and load shedding [6]. This reduces the required CPU resources and in some cases the memory requirements (e.g., if a time-based sliding window is used). Filtering is another method that can be used to change aspect $A1$, for example, using outlier or burst detection methods. Filtering may change the range and distribution of values, and thus may impact the memory requirements of the synopsis.

A2: Input parameters. Most stream-mining algorithms have input parameters that influence how well their output approximates the actual mining result (cf. Section 4.2). Consequently, these input parameters are one of the main aspects that determine the trade-off between resource requirements and output quality. A better approximation of the mining result implies more runtime per stream element and more main memory required to store the synopsis. Consequently, changing the input parameters of the online component directly influences its CPU and memory requirements. The CluStream algorithm uses an input parameter q to set the number of micro-clusters into which the data stream is partitioned. A higher value of q results in higher memory requirements, as each micro-cluster needs to be stored along with its history. Likewise, q influences the CPU requirements, because for each new element of the data stream q different micro-clusters have to be considered to find the best matching one.

A3: Query parameters. User queries, particularly the query parameters, and, in the case of continuous queries, the frequency of result updates influence an algorithm's resource requirements as well. In particular, the offline mining component is affected. The larger the part of the synopsis that needs to be accessed, the more CPU cycles will be required. If data mining is conducted directly on the sensors in a wireless network, the bandwidth requirements are also influenced

Q_M	Methodical quality	Q_T	Temporal quality
Q_{Ma}	Quality of approximation	Q_{Tr}	Time retrospect
Q_{Mi}	Qualitiy of interestingness	Q_{Tg}	Time granularity
		Q_{Tc}	Reaction time

Table 1: Classes of quality measures

by the the size of the mining result (e.g., the number of output clusters for CluStream) and the frequency of result transmission. Adapting query parameters should only be done if other resource adaptation methods are not sufficient for meeting the given constraints, as it affects the usefulness and interestingness of results to user queries significantly.

Note that the list of resources affected by the above aspects is exemplary rather than exhaustive. Changes to any of them usually influence more than one resource at the same time and more than one quality measure (cf. Section 4.2). For example, decreasing q in CluStream will speed up the clustering algorithm and also decrease its memory requirements. As presented in Section 3, the three aspects were previously identified as algorithm granularity settings [12]. However, this classification is not as general as our framework. For example, algorithm processing granularity (APG) only refers to adapting the CPU requirements, while the influence on memory consumption is not considered. Further, the authors rely on randomization and approximation techniques to adapt the APG and do not consider to adapt the existing input parameters of the algorithm.

4.2 Quality Measures

The quality of stream mining describes how well the actual mining results, i.e., results that could be achieved when unlimited resources were available, are approximated. Aspects $A1$ through $A3$, which were detailed in Section 4.1, impact the resource requirements *and* the achieved quality. To assess the mining quality, we use quality measures based on the described parameters. In previous work [10] we identified different classes of quality measures, which are summarized in Table 1. This classification is comprehensive, yet extensible without restricting the proposed model.

The quality $\mathcal{Q}$ of stream mining can be classified into methodical quality Q_M and temporal quality Q_T. The different measures in Q_T are identical for all mining tasks, while Q_{M*} represents classes of quality measures that are specific to the data-mining task and algorithm. For further details, we refer the reader to [10].

Similar to constraints on resources, upper and lower bounds can be defined for all quality measures $Q \in \mathcal{Q}$. Specifically, for a quality measure $Q \in \mathcal{Q}$, $Q^{\perp}$ and $Q^{\top}$ denote the minimum and maximum quality, respectively, that needs to be achieved. The desired quality has a strong influence on the amount of resources the stream-mining algorithm requires. In general, the higher the quality should be, i.e., the better the approximation, the more resources are required.

Sometimes, more than one quality measure is influenced at the same time. In CluStream, different numbers k of output clusters provide different levels of interestingness (i.e., different values for Q_{Mi}) depending on the number of natural clusters. The value of k also influences Q_{Ma}, as more clusters represent a more fine-grained approximation of the stream.

Example 4.1 *Below we list all parameters of CluStream and which class of quality measures they influence.*

q **influences** Q_{Ma} *A higher number of micro-clusters approximates the stream elements more accurately.*

α **influences** Q_{Tg} *The more frequently snapshots of the micro-clusters are stored in the PTF, the better is the temporal granularity at which they can be accessed.*

l **influences** Q_{Tg} *The more snapshots are stored for each level of the PTF, the higher is the temporal precision when answering user queries. This is also related to the effect of parameter α and is discussed in [1] in more detail.*

h **influences** Q_{Tg} *The queried time horizon h determines the temporal granularity at which snapshots can be accessed, as older snapshots are stored at a coarser granularity.*

k **influences** Q_{Ma} **and** Q_{Mi} *See above.*

We define two quality measures based on CluStream's parameters:

$$Q_1 = q/k \quad \text{and} \quad Q_2 = \alpha^l$$

Q_1 was already used to evaluate the clustering quality of the original CluStream algorithm in [1]. The authors found that high-quality clustering results can be achieved for a q/k ratio of about 10. We select a second quality measure $Q_2 = \alpha^l$ because it represents the temporal quality in CluStream. As described in Section 2, the number of snapshots that are stored in each level of the PTF is limited to $\alpha^l + 1$, and the approximation of the queried time interval also depends on the value of α^l.

5. PARAMETER ADAPTATION

Given the characteristics of stream-mining algorithms introduced in the previous section, the goal of a stream-mining process can be stated as follows:

Mine the data stream continuously with the maximum possible quality, such that each quality measure $Q \in \mathcal{Q}$ is within its constraints $Q^{\perp}$. Input parameters that determine the quality are subject to dynamic resource constraints and have to be chosen such that the requirements for each resource $R \in \mathcal{R}$ are within its constraints $R^{\perp}$.

In this section, we present a framework to model and achieve this goal. For this, we first introduce *adaptation factors*, i.e., parameters that are used to adapt the resource requirements of an algorithm as well as the quality in Section 5.1. We then define a set of functions to model the correlation of parameters, quality, resource requirements, and other variables in the stream-mining process in Section 5.2. In Section 5.3, we show that the goal stated above can be expressed as an optimization problem and propose a solution to this problem. Then, in Section 5.4, we list requirements for algorithms to be used within the proposed framework. Finally, in Section 5.5, we outline how to apply our framework to a frequent itemset mining algorithm that fulfills these requirements.

5.1 Adaptation Factors

As described in Section 4, a lot of variables, e.g., the stream rate and input parameters of the mining algorithm, are present in data-stream mining. Constraints $\mathcal{R}^{\perp}$, $\mathcal{R}^{\top}$, $\mathcal{Q}^{\perp}$, and $\mathcal{Q}^{\top}$ are also variables, as they are set by users or imposed by the stream-mining system (e.g., the maximum amount of main memory $mem^{\top}$ that can be used by an algorithm). We categorize all variables into *parameters* $P \in \mathcal{P}$ and *observations* $O \in \mathcal{O}$, with $\mathcal{P} \cap \mathcal{O} = \emptyset$. Parameters P are variables that can be set dynamically, whereas observations O

are variables that can only be observed. Some observations can be influenced by changing parameter values. For example, one cannot directly influence the stream rate of the raw data stream (hence, it is an observation), but application of sampling or load shedding will influence it.
Variables in data stream mining are not independent of each other. In most cases, a change of one variable's value causes other variables to change as well. For example, if $q \in \mathcal{P}$, the number of micro-clusters in CluStream, is changed, the number of required CPU cycles, main memory requirements, and methodical quality Q_M change as well. Thus, parameters are "tuning knobs" for the data stream-mining process, as they influence resource requirements and quality. In our model, some or all of the parameters in $\mathcal{P}$ are selected as *adaptation factors*, i.e., parameters that are automatically adjusted in order to adapt an algorithm's resource requirements and the achieved quality. Parameters not chosen as adaptation factors remain constant throughout the lifetime of the stream-mining process. Thus, in the following we only denote adaptation factors as parameters. Consequently, if not all parameters in $\mathcal{P}$ are adaptation factors, the remaining ones are in the set $\mathcal{O}$ of observations (i.e., they are not automatically adjusted).

Example 5.1 *For the CluStream algorithm, we chose two adaptation factors q and l, i.e., $\mathcal{P} = \{q, l\}$. Although α is a parameter as well, it cannot be changed dynamically during the stream analysis as most snapshots maintained in the PTF would become invalid and much of the historical information about data stream would be lost. Thus, $\alpha \in \mathcal{O}$. In addition, parameters h and k remain constant as well, as we do not want to adapt the query parameters. Note that, however, the adaptation of query parameters may be beneficial for other algorithms and is certainly supported by our model.*
As the PTF increases in size as time progresses, variable t, the number of elapsed time units, is also an observation. In summary, CluStream's parameters and observations are as follows:

$$\mathcal{P} = \{q, l\} \quad \text{and} \quad \mathcal{O} = \{\alpha, h, k, t\}$$

5.2 Correlation Model

We published a preliminary version of the model presented in this section in [15], which did not consider parameter adaptation as a multiobjective optimization problem. At any given time t, the *system state* of a stream mining algorithm is defined as the values of all its parameters and observations. To indicate values of variables at a specific point in time, an index is added, e.g., $mem_t^{\top}$ denotes the maximum amount of available memory at time t.
For a given stream mining algorithm, a function φ can be defined that, given the system state at time t, determines the resource requirements for a resource $R \in \mathcal{R}$ at that time:

$$\varphi : \mathcal{P}_t \times \mathcal{O}_t \rightarrow \mathcal{R}_t$$

Function φ defines how to compute the resource requirements of an algorithm based on values for parameters $\mathcal{P}$ and observations $\mathcal{O}$. A separate function is defined for each resource $R \in \mathcal{R}$.

Example 5.2 *The signature of φ for CluStream's main memory requirements $mem \in \mathcal{R}$ is:*

$$\varphi : \{q_t, l_t\} \times \{\alpha_t, t\} \rightarrow mem_t$$

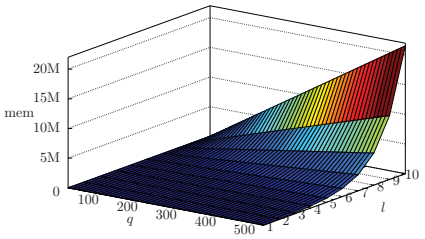

Figure 2: Main memory requirements of CluStream

CluStream's memory requirements are composed of two parts, the size of the PTF and the amount of memory consumed by the current q micro-clusters. As we derived in [10], the overall memory requirement is given by the following equation and is measured in floating-point numbers:

$$\varphi : \quad mem(q_t, l_t, \alpha_t, t) = q_t(2d + 1 + c_{LRU})$$
$$(q_t(2d + 1) + 1) \cdot (\alpha_t^{l_t} + 1) \cdot \lceil log_{\alpha_t}(t) \rceil$$

In the above equation, c_{LRU} is a constant and d is the (constant) dimensionality of objects in the data stream.
Figure 2 depicts function φ for mem against parameters q and l in million floating-point numbers ($c_{LRU} = 20$, $d = 2$). The figure depicts the memory requirements for $t = 1,000,000$ and $\alpha = 2$. For the sake of presentation, we limit the values of q and l in Figure 2 (and also in subsequent figures) to the intervals $q \in [1, 500]$ and $l \in [1, 10]$.

Using φ, optimal parameter values can be determined to satisfy resource constraints. Observations $\mathcal{O}$ are either taken from the current system state or represent predictions (see Section 7). Similar to φ, we define a separate function ψ for each quality measure $Q \in \mathcal{Q}$ based on the system state at time t:

$$\psi : \mathcal{P}_t \times \mathcal{O}_t \rightarrow \mathcal{Q}_t$$

Example 5.3 *We defined two quality measures Q_1 and Q_2 for CluStream. The corresponding functions ψ are as follows:*

$$\psi_1 : Q_1(q_t, k_t) = q_t / k_t$$
$$\psi_2 : Q_2(\alpha_t, l_t) = \alpha_t^{l_t}$$

As Q_1 grows linearly with q, and Q_2 grows exponentially with l, each quality measure depends on one variable.

By evaluating φ and ψ using the same parameter and observation values, we obtain a mapping that assigns each set of quality values to an amount of resources required to achieve these values. Each of these possible combinations is called a *configuration*: a tuple of values for observations, parameters, qualities, and resource requirements.

Example 5.4 *Figure 3 depicts the mapping of Q_1 and Q_2 (shown on a log-scale) to mem for CluStream. Each point in Figure 3 corresponds to a configuration.*

To keep track of previous values of observations and parameters, we introduce sets $\mathcal{P}_T$ and $\mathcal{O}_T$, so called *timelined variable values*. They are needed to compute the accumulated quality of a user-queried time interval. Distinct time intervals $[t_i, t_{i+i})$ can be defined during which variable values did not change. For each variable, the concatenation of all time intervals represents the complete history of this

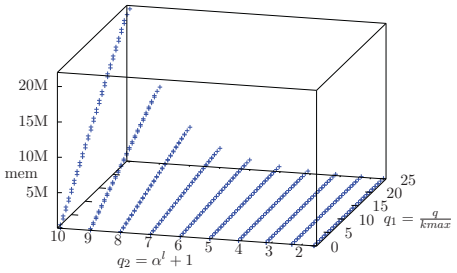

Figure 3: Mapping quality values for Q_1 and Q_2 to main memory requirements of the CluStream algorithm.

variable's values. For each parameter $P \in \mathcal{P}$, timelined parameter values P_T are defined as follows:

$$P_T = \{P_{t_i} | P_{t_i} \text{ was effective during time interval } [t_i, t_{i+i})\}$$

$\mathcal{P}_T$ contains the timelined variable values for all $P \in \mathcal{P}$. Set $\mathcal{O}_T$ is defined analogously. $\mathcal{P}_T$ and $\mathcal{O}_T$ are input for function(s) ψ to compute timelined quality values $\mathcal{Q}_T$. These values are used to support time sensitivity in situations where different quality values were effective during the queried time interval. To determine the quality $Q_{[t_i,t_j]}$ for a queried time interval $[t_i, t_j]$, we define a function ξ for each quality measure $Q \in \mathcal{Q}$:

$$\xi : \mathcal{Q}_T \times t_i \times t_j \rightarrow \mathcal{Q}_{[t_i,t_j]}$$

Example 5.5 *Consider quality measure $Q_1 = q/k$ and a user query for the clustering results of time interval $[t_i, t_j]$. Within this time interval, several different values of q might have been used in the online clustering, resulting in quality values Q_{1,t_i}, $Q_{1,t_{i+1}}$, etc. To determine the quality of the mining result, function ξ is defined as the minimum of all values of Q_{1,t_k} that have been effective:*

$$\xi : Q_{1,[t_i,t_j]}(Q_{1,T}, t_i, t_j) = min_{Q_{1,t_k} \in Q_{1,T} : t_i \leq t_k \leq t_j}(Q_{1,t_k})$$

Storing timelined variable values requires additional memory. The amount of memory required should be reduced by merging variable values in the timeline as soon as a quality measure is decreased at some point in time. In the CluStream example, as soon as l is decreased, snapshots on each level are deleted so that only the $\alpha^l + q$ most recent ones are maintained. As the deleted snapshots can never be recovered, the timeline for parameter l can be deleted as well. Note that for special adaptivity schedules based on forecasting resource consumption (cf. Section 7), the whole timeline of parameter values might be required. However, usually it is feasible to limit the number of previous variable values that are stored, e.g., to the n most recent variable values.

5.3 Optimization Problem

The above stated goal of a stream-mining process is an optimization problem, as the objective is to maximize quality subject to resource and quality constraints. As mentioned, constraints $\mathcal{R}^{\perp}$ and $\mathcal{Q}^{\perp}$ might not exist for all resources and quality measures. However, we can assume that at least one resource $R \in \mathcal{R}$ has an upper limit $R^{\top}$, since otherwise resource awareness would not be necessary. Thus, there are multiple objectives, namely to keep resource requirements below $R^{\top}$ and to maximize all quality measures. Such an optimization problem where several (conflicting) objective functions exist, is called a *multiobjective optimization problem*, and has been discussed, e.g., by Ehrgott in [8]. The

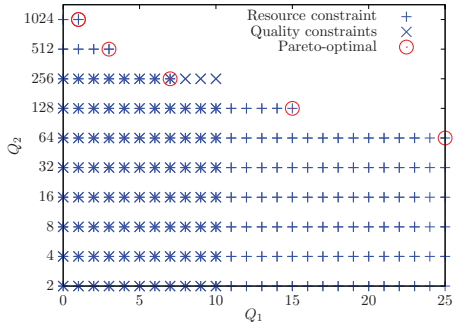

Figure 4: Possible configurations in the presence of resource constraint $mem^{\top} = 4M$, or quality constraints $Q_1^{\top} = 10$ and $Q_2^{\top} = 500$.

optimization problem in our context contains one objective function for each quality measure and each resource, defined by functions φ and ψ. It can be decided for each $R \in \mathcal{R}$ and $Q \in \mathcal{Q}$ independently if it should be maximized, minimized, or if it should be constrained by upper or lower bounds. For example, on sensor nodes battery usage should usually be minimized. In contrast, memory consumption should just be constrained, as there is no benefit from using less memory than available.

Given a system state at time t, a solution to the multiobjective optimization problem yields an optimal configuration. We discuss details of how to solve the multiobjective optimization problem in Section 6. The parameter values in the optimal configuration are immediately used as the new parameter values of the stream-mining algorithm (cf. Section 7). Constraints limit the number of possible configurations. Specifically, if one or more constraint for $\mathcal{R}^{\perp}$, $\mathcal{R}^{\top}$, $\mathcal{Q}^{\perp}$, and $\mathcal{Q}^{\top}$ is set, some configurations become invalid. Additionally, constraints τ can be defined on one or more quality measures, resources, or a combination of both, to further limit the set of possible configurations.

Example 5.6 *For our running example, we choose to maximize Q_1 and Q_2 and to apply a constraint $mem^{\top} = 4M$, i.e., four million floating-point numbers. The resulting possible configurations are depicted in Figure 4 and are marked by '+'. In the figure, only quality measures Q_1 and Q_2 are illustrated, as the respective parameter values for each configuration can be obtained from function ψ. Note that Q_2 is again shown on a log-scale. Also note that, for clarity of presentation, not all possible configurations are shown, but only those for which Q_1 is an integer. For example, $q = 50$ yields $Q_1 = 42/20 = 2.1$, but only $Q_1 = 2$ and $Q_1 = 3$ are shown. Figure 4 also depicts possible configurations (marked by an 'x') when adding constraints $\mathcal{Q}^{\top}$ for both quality measures, namely $Q_1^{\top} = 10$ and $Q_2^{\top} = 500$. Further, it may be beneficial to balance both considered quality measures. This can be achieved by adding a constraint τ, e.g., like the following:*

$$\tau : \log_2(Q_2)/Q_1 - 1 \leq 0$$

Now that we introduced all components, we define the multiobjective optimization problem that needs to be solved in

order to determine an optimal configuration as follows:

$$\begin{aligned} \text{minimize} \quad & \varphi, \ -\psi \\ \text{subject to} \quad & \varphi - \mathcal{R}^{\top} \leq 0 \\ & \mathcal{R}^{\perp} - \varphi \leq 0 \\ & \psi - \mathcal{Q}^{\top} \leq 0 \\ & \mathcal{Q}^{\perp} - \psi \leq 0 \\ & \text{additional constraints } \tau \end{aligned}$$

Note that objective functions ψ are maximized by minimizing $-\psi$. This is a common technique when formulating optimization problems.

5.4 Requirements for Algorithms

The proposed model can only be applied to stream-mining algorithms that fulfill the requirements listed below.

Instantiate functions of correlation model In order to apply our model for quality-driven resource adaptivity, functions φ, ψ, and ξ need to be devised for the stream mining algorithm in question. The functions can be derived either from an analysis of the algorithm, as we did for CluStream, or from a set of test runs. Obviously, the devised functions have to be defined on compatible domains. An intuitive approach is to define them on integer or real values, but soft string values, such as "good" and "poor", can also be chosen.

Query parameters In order to influence an algorithm's resource requirements for the offline mining component, the algorithm needs to employ query parameters.

Locality of parameter effects It is desirable that the synopsis is partitioned into temporally independent sections. This is important to ensure that different values of parameters only have a "local" effect on the quality.

Example 5.7 *To explain the concept of local effects of parameters, imagine that at some point in time the value of q in CluStream is increased from q_1 to q_2. Snapshots taken when q_1 was effective contain exactly q_1 micro-clusters, while snapshots after the parameter adaptation contain q_2 micro-clusters. Thus, when a user-queried time interval only contains snapshots taken after q_2 became effective, then the lower quality that resulted from parameter q_1 does not affect the result quality of this user query.*

Consequently, it should also be possible to query each of the independent sections in the synopsis separately. Otherwise, the lowest quality ever used determines the quality of the overall mining result. Another way to achieve locality is to choose a synopsis where information expires after a certain amount of time (e.g., like in certain window models). Note that locality of parameter effects is not a strict requirement. The proposed model can be applied in any case, but the algorithm will not be able to recover from parameter settings that cause low quality.

5.5 Application to Frequent Itemset Mining

To demonstrate the broad applicability of our model, we now outline how it can be applied to another fundamental data mining problem, the mining of frequent itemsets. A popular stream-mining algorithm for this problem is proposed in [13]. The algorithm uses an FP-tree like data structure called *pattern tree* as synopsis. A *tilted time window* table, similar to CluStream's PTF, is associated with each node in the tree. In this table, the frequencies of the itemset in different time intervals are stored during the online mining phase. In the offline mining phase, an FP-growth algorithm is run on this synopsis to extract frequent itemsets according to the *query parameters* (class A3 cf. Section 4), such as the queried time window. The algorithm from [13] applies some approximations that are typical for frequent itemset algorithms on data streams. First, the true support σ_e of a mined itemset e is approximated using a user-defined parameter $\epsilon \in (0, 1)$, resulting in an approximate support $\hat{\sigma}_e$ for which the following inequality holds:

$$\sigma_e - \epsilon \leq \hat{\sigma}_e \leq \sigma_e \tag{1}$$

Therefore, ϵ represents an *input parameter* from class A2 (cf. Section 4) that influences the quality of approximation Q_{Ma}. As the pattern tree holds only itemsets having a support of at least ϵ, the value of this parameter directly affects the number of nodes in the tree, and thus the resource requirements. The second aspect concerns the time granularity for which frequent itemsets are determined. A user-defined *input parameter* b determines the size of the windows to process, and thus the time granularity Q_{Tg} at which mining results can be obtained. Note that this is similar for algorithms using sliding windows, while the rather seldomly used landmark windows are not well suited for our model as the requirement for locality of parameter effects is not fulfilled. Besides the above mentioned queried time window, the queried minimal support of an itemset represents another *query parameter* from class A3, influencing Q_{Mi}.

Consequently, ϵ and b act as adaptation factors for our model. Both parameters influence the memory and CPU requirements as well as the mining quality. In every iteration, the algorithm processes a batch of b transactions and finds itemsets having at least support ϵ. Only those are stored in the pattern tree. Thus, the value of b affects the number of entries in the tilted time window table, because for larger batches entries are made less frequently. Additionally, itemsets that appear as frequent only in smaller windows will not be added with larger values of b.

Necessary *observations* in the context of frequent itemset mining include various stream properties (class A1 cf. Section 4), such as average length of transactions, total number of items in the stream, and the distribution of the items in the stream. Such properties can, for example, be inferred from a sample of the stream. Note that inaccurate values for these observations will not invalidate our model but only lead to less accurate resource adaptation.

Having identified parameters and observations from the different classes, the remaining task is to define the functions used in the correlation model. This is out of scope for this paper. [10] provides a basis for this by presenting formulas for memory consumption and output quality based on the above discussed parameters.

Note that the frequent itemset stream mining algorithm in [13], similar to many other such algorithms, provides *quality guarantees*, i.e., there are no false negatives and the true support of itemsets in the result set is at most ϵ lower than the reported frequency (cf. inequality 1). A remarkable benefit of our model is that the resource-adaptive algorithm will still provide guaranteed error bounds in the same way as the original algorithm did. Our parameter adaptation will vary the algorithm's error bound, i.e., parameter ϵ, accord-

ing to the available resources. However, by monitoring these changes it will always be possible to compute the exact value of the error bounds for the computed frequent itemsets.

6. FINDING OPTIMAL PARAMETERS

In this section, we describe an approach to find the solution for the introduced multiobjective optimization problem, which corresponds to finding the optimal configuration. First, we discuss how to find possible configurations in Section 6.1. In Section 6.2, we then use the notion of Pareto-optimality to determine a set of optimal configurations, where each configuration is optimal with respect to at least one of the objectives. Finally, in Section 6.3, we discuss how a single optimal configuration can be determined.

A good overview of methods to solve multiobjective optimization problems, including benefits and drawbacks of each method, is provided by Freitas in [11]. The work focuses on the application of multiobjective optimization in data mining. Other survey papers that present different methods are, e.g., by Coello [7] and Lamont [19].

6.1 Finding Possible Configurations

Recall Example 5.6, where constraints $\mathcal{Q}^\top$ on the quality measures of CluStream were set in order to limit the number of possible configurations. Setting constraints $Q_1^\top = 10$ and $Q_2^\top = 500$ results in about 1800 possible configurations for the parameter settings of CluStream. However, the number of possible configurations is finite only because parameters q and l are integers. Considering algorithms that have real-valued parameters, there is an infinite number of possible configurations, and hence the search space is infinite as well. In this case, parameter adaptation can be extremely inefficient, if not impossible at all. We therefore propose to associate a *step size* P_Δ with each parameter $P \in \mathcal{P}$ that is not an integer, to indicate the minimum increment or decrement of this parameter. This limits the number of possible configurations and ensures that parameter adaptation is only done when significant improvements can be achieved. The step size P_Δ should be chosen such that the tradeoff between the accuracy with which parameter P can be adapted and the computational effort to find the optimal value of P is acceptable. Naturally, a small step size results in a high number of possible configurations from which the optimal one will be chosen. But, the given resource and quality constraints can be met more accurately if a high number of possible configurations exist. Detailed considerations about appropriate values for P_Δ are not within the scope of this paper.

As we will describe in Section 7, the resource adaptivity component itself helps to limit the number of possible configurations further. Specifically, an indicator helps to determine the reason for activating the parameter adaptation, i.e., if quality measures are not met, resources are underutilized, or resource limits exceeded. This information helps to limit the range of parameter values to consider.

Example 6.1 *Assume that CluStream's current parameter values are $q = 400$ and $l = 8$. At some point in time, constraint $mem^\top$ is increased, and thus a new optimal configuration is needed. As function φ for CluStream is monotonically increasing (cf. Figure 2), configurations with $q < 400$ and $l < 8$ are no candidates for the new optimal configuration. Thus, possible configurations are limited to those where either $q > 400$ or $l > 8$ (or both) is true. Of course, configurations having $q < 400$ and $l < 8$ are technically still in the set of possible configurations. To formalize this limitation, we add a constraint τ, which is valid only for the current cycle of the resource adaptivity component, to prevent configurations having $q < 400$ and $l < 8$ from being used during this particular pass of parameter adaptation.*

A naive approach to find the optimal configuration is to first find all possible configurations and then choose one from this set. This can be done by enumerating all possible parameter values until the corresponding resource consumption exceeds constraints $\mathcal{R}^\top$ or $\mathcal{Q}^\top$ and falls below $\mathcal{R}^\perp$ or $\mathcal{Q}^\perp$. This is easy to do, because functions φ and ψ usually do not have local maxima, but are monotonically increasing. However, depending on the step size, domain of parameter values, and the values of constraints $\mathcal{R}^\top$ or $\mathcal{Q}^\top$, the number of possible configurations may be quite large and thus expensive to compute. In the next subsections, we describe alternative methods to find the optimal configuration.

6.2 A Skyline Approach

Many of the possible configurations do not need to be considered, as other configurations exist that *dominate* them. A configuration c_1 is not dominated by any other possible configuration c_2 if c_1 is better with respect to at least one objective, e.g., c_1 has lower resource requirements or a higher value for at least one quality measure. Such non-dominated configurations are called *Pareto-optimal* and are contained in the Pareto front of all possible configurations. In the database community, the Pareto front of a discrete set is called a *skyline* [5]. For parameter adaptation, only Pareto-optimal configurations need to be considered.

Example 6.2 *Recall Example 5.6 and the associated Figure 4. For configurations marked by a '+', a resource constraint $mem^\top = 4M$ was applied. Out of these configurations, all Pareto-optimal configurations are highlighted by a (red) circle in Figure 4. Each Pareto-optimal configuration maximizes at least one of the quality measures while obeying the resource constraint $mem^\top$. Note that for the other example illustrated in Figure 4, where both quality measures are constrained, only one configuration is Pareto-optimal, i.e., the one where $Q_1 = 10$ and $W_2 = 256$.*

Different algorithms exist to determine the Pareto front of a multiobjective optimization problem. A survey of existing methods was published in, e.g., [7; 19]. A popular approach is to use evolutionary algorithms. In the context of our model, the Pareto front can often be determined analytically. This is because functions φ and ψ are monotonically increasing, and thus it is straightforward to narrow down the parameter settings for an optimal configuration.

Example 6.3 *Finding Pareto-optimal configurations for CluStream analytically is very efficient. Assume that at a given point in time, observations are $k = 20$, $\alpha = 2$, and $t = 1,000,000$ (again, $d = 2$ and $c_{LRU} = 20$). We set a constraint $mem^\top = 4M$. Our goal is to find the Pareto-optimal configurations for this setting, specifically parameter values for q and l such that both quality measures Q_1 and Q_2 are optimal with respect to the resource constraint $mem^\top$.*

We observe that the number of possible values for parameter l is very limited, i.e., $l = 30$ already yields up to 1

billion snapshots per level in the PTF, which (heuristically) is sufficient in most cases. As there is only one Pareto-optimal configuration for each value of l, the corresponding value of q can be computed using a transpose of function φ. Specifically, for each value of $l \in [1, 30]$ we compute the corresponding maximum *value of q such that the resource constraint $mem^{\top}$ is obeyed. The Pareto-optimal value of q, given a value of l as well as observations and constants mentioned above, is computed as follows:*

$$q = \left\lfloor \frac{4,000,000 - \log t - 2^l \cdot \log t}{30 \log t + 5 \cdot 2^l \cdot \log t} \right\rfloor$$

6.3 Choosing the Optimal Configuration

Each of the configurations in the Pareto front is optimal with respect to at least one objective. Different approaches can be used to decide which of these configurations is actually chosen. In this section, we discuss three such approaches.

Weights for all objectives One common approach is to assign a weight to each objective and subsequently combine all objectives in one objective function. The sum of all weights usually equals one. The single objective function that is created this way yields a unique solution, which can be computed without determining the Pareto front of all possible configurations. However, choosing the weights for all objectives is hard, as pointed out by Freitas in [11]. We therefore do not recommend to employ this approach unless it is clear how to weight the objectives.

Least amount of unused resources If $R^{\top}$ exists for at least one resource, we can choose one configuration from the Pareto front for which the resource requirements computed using function φ are as close to $R^{\top}$ as possible. That way, the optimal configuration is chosen such that the amount of unused resources is minimized. If constraints exist for more than one resource, one could aim at minimizing the sum, or the sum of squares, of unused resources. In case two or more configurations are equally good in terms of resource utilization, we either choose one at random or follow the approach described below.

Fewest parameter changes A third approach to select the optimal configuration is to choose the one that causes the least parameter changes. That is, given the current parameter settings of the stream-mining algorithm, we select the new configuration such that either as few as possible parameters change, or that the changes that have to be applied to all parameters are as small as possible. Fewer changes to parameters are beneficial for the overall runtime overhead imposed by parameter adaptation. For example, in CluStream, decreasing the value of q from q_1 to q_2 means that we have to merge $q_1 - q_2$ micro-clusters. The overhead decreases with a decreasing difference between q_1 and q_2.

The three approaches above are just examples of methods to choose an optimal configuration. Of course, other approaches are conceivable. Typically, the decision how the optimal configuration should be chosen depends in the specific application.

Generally, solving a multiobjective optimization problem can become quite expensive. The runtime of the whole adaptation process is dominated by this. However, the properties of the objective functions, such as their monotonicity, help to implement efficient approaches. For the example of CluStream, we can find a solution with only few computations. As properties of the objective functions may vary for different stream-mining algorithms, it is not possible to provide general rules for the cost associated with this step. As a general guideline, objective functions with according properties and approximating algorithms for solving the optimization problem should be preferred.

7. TRIGGERING ADAPTATION

The initial parameter values are chosen based on initial observations, either based on observing the stream for a short period in the beginning or by assuming values for each of the observations. They have to be chosen such that the initial resource requirements do not exceed the constraints $\mathcal{R}^{\top}$. Then, there are two aspects of the actual scheduling of parameter adaptation: *(1)* decide whether to trigger the component, and *(2)* decide when to evaluate that decision again. For question *(1)*, we use a function ρ. ρ is used to decide whether parameters have to be adapted and further acts as a guideline for the efficient solution of the optimization problem. We suggest a simple encoding for ρ:

$$\rho(\mathcal{P}_T, \mathcal{O}_T, \mathcal{Q}^{\perp\!\!\!\perp}, \mathcal{R}^{\perp\!\!\!\perp}) := \begin{cases} 0, & \text{if no adaptation is required} \\ 2, & \text{if resources too low} \\ 3, & \text{if quality too low} \\ 4, & \text{if resources too high} \\ 5, & \text{if quality too high} \end{cases}$$

Value $\rho = 2$ indicates that either the resource requirements are below $R^{\perp}$ for at least one resource $R \in \mathcal{R}$, or that at least one $R \in \mathcal{R}$ is underutilized with respect to $R^{\top}$. The same applies for 3 and the quality limits, and in analogy for 4 and 5. If quality and resource limits are affected, ρ returns the product of the according values, e.g., $2 \cdot 3 = 6$ if both, resources and quality, are too low. ρ makes use of functions φ and ψ to compute resource utilization and quality values. In all cases where $\rho > 0$, the optimization problem has to be solved. If a solution cannot be found in the current state, the failed parameter adaptation is signalized and reaction is left to the system or user. In most cases this is due to $\mathcal{R}^{\top}$ set too low or $\mathcal{Q}^{\perp}$ set too high, which can be indicated by the adaptivity component. Consequently, the system or user will likely react by adjusting resource or quality constraints, or by terminating the stream-mining process.

When new parameter values are set in the stream mining algorithm, some adjustments may have to be applied to the synopsis. This is the case if at least one of the new parameter values decreases the quality or resource requirements of the stream-mining algorithm. Then, elements in the synopsis need to be merged or pruned according to the algorithm's build-in routines. In CluStream, if the number q of micro-clusters is decreased, micro-clusters need to be merged until only q of them are left. The used routine is also frequently used during regular stream processing. Due to their frequent use, such merging and pruning techniques are designed to be very efficient. Thus, adjusting the synopsis to the newly set parameter values is done very efficiently as well. However, it is still beneficial to keep the changes of parameters during parameter adaptation to a minimum.

We assume that the underlying stream system activates the resource adaptivity component as soon as the resource limits change. However, exhausted limits may result in a system failure. Thus, a crucial question is when to trigger the adaptivity component while the limits are stable. To prevent a

violation of the limits, the adaptivity component should be triggered preventively. One can choose to adapt parameters more prudently, e.g., by setting the effective $R^\top$ to 80% of the amount of resources that are actually available. This decreases the quality that can be achieved. It is comparable to the approach chosen in [10]. The authors use a filling factor f that reflects the percentage of available resources already consumed. If f is above or below certain thresholds, parameter adaptation is triggered. An alternative approach was proposed in [12], which calculates the number of time steps remaining until the resources are exhausted. When this is predicted to happen before the adaptivity component will be triggered again, it has to be triggered in the current time step. This relates to the answer for question *(2)* from above, which is focus of the following subsection.

7.1 Adaptation Schedules

When the adaptation component was triggered, it has to be decided when it should be triggered again. There is a tradeoff between the overhead imposed by the additional computations and the benefits of frequently triggering the resource adaptivity component (e.g., the ability to quickly react to changing resource requirements). We propose two different approaches. Both approaches cannot guarantee that the adaptation will prevent the algorithm from failure due to exhausted resources. Especially in scenarios with frequent and drastic changes in the observations, appropriate and timely predictions for the algorithm's resource consumption are hard to achieve.

In Situ Adaptivity. We call the first approach *in situ adaptivity* because it acts based on the situation at the current point in time. The adaptivity component is activated every δ time units. The value of the variable δ is fixed throughout the stream mining process. It is therefore an additional observation, i.e., $\delta \in \mathcal{O}$.

The advantage of this approach is that there is no need to predict the future system state. Thus, it is a simple and easily implemented method to achieve efficient resource utilization in most scenarios. But, in situ adaptivity may not be flexible enough to suitably react to changes in the observations. As it does not anticipate future observations, it might fail to act appropriately in situations where there are major changes in observations that result in greatly increased or decreased resource requirements. It is therefore best suited for scenarios where observations have fairly stable values. Using a filling factor as described above decreases the risk that resources are exhausted due to unexpected changes of observations, while preserving the simplicity and convenience of the in situ adaptivity approach. Besides the problem of possible failures before δ time steps have passed, underutilized resources or violated quality constraints might result in a sub-optimal analysis for up to δ time units.

For our running example, we use an in situ adaptivity schedule to apply resource adaptivity to the CluStream algorithm. Details about the used schedule as well as an appropriate function ρ are described in Section 7.2.

Proactive Adaptivity. Alternatively, a prediction for future resource requirements can be used to determine for how long the current parameter values will be valid. The adaptivity component is activated after the predicted time span has expired. We call this approach *proactive adaptivity.*

Timelined variable values can be used to predict observations and the parameter adaptation can be based on this anticipated system state, rather than on the current system state. That means, functions φ and ψ take predicted observations $O_{t+\triangle}$ as input rather than O_t. In addition, predicted observations can be used to decide about the validity time span of the current parameter values as described above. The prediction of observations may reveal changes of observations that necessitate parameter adaptation, which is not possible with the in situ approach. Finally, the prediction is used to compute the validity time span δ for the new parameter values. Unlike in the in situ approach, the value of δ is not constant. In the most straightforward case, δ is set to $\delta = \triangle$. A longer validity time span is possible if the predicted trend of the observations indicates that all constraints can be met for a longer time. Such a validity time span can also be predicted in the in situ approach, but this would be restricted to using only the current system state. Because the value of the variable δ changes throughout the stream mining process, it is treated as an additional parameter, i.e., $\delta \in \mathcal{P}$.

Note that it is possible to make parameter values effective in the future, i.e., $t + \gamma$. This can be achieved by setting $\delta = \gamma$ and $\rho = 0$, thus omitting parameter adaptation at the current time. New parameter values are computed at time $t + \gamma$ where the predictions are based on more recent observations. Typically, $\gamma < \triangle$ holds, as observations are predicted for time $t + \triangle$. If parameter adaptation is necessary, the new parameter values need to be effective before time $t+\triangle$ in order to meet resource and quality constraints.

A proactive approach is more complex and generates more runtime overhead than in situ adaptivity. It should therefore only be used if observations exhibit fairly erratic changes, which makes in situ adaptivity not appropriate. Proactive adaptivity can benefit from observing such changes and according patterns over time. The critical part is the actual prediction of observations. If it is inaccurate, the algorithm might crash in the worst case, because resources are exhausted. Any technique for predicting observations has to be a fairly lightweight process compared to the actual stream mining algorithm. The complexity of the computation of the anticipated system state depends on the employed method as well as on the number of previous variable values taken into account. A fairly lightweight but effective prediction can be done using linear regression. More elaborate prediction schemes may incorporate methods for periodicity estimation or burst detection, which could be especially beneficial to detect trends in the stream rate as it is one of the most crucial observations. The proactive adaptivity approach promises to be especially useful with respect to the additional overhead that the whole resource adaption implies. This is because we may be able to predict when the number of required CPU cycles will exceed the specified resource constraint, and thus appropriate parameter adaptation can take place before the resource becomes exhausted. Without such a prediction, the parameter adaptation my have to take place during the time when CPU cycles are already a scarce resource.

7.2 Adaptivity Schedule for CluStream

For our resource-adaptive version of CluStream, an in situ adaptivity schedule is used. We omit the algorithm here, as

it implements the function ρ introduced above in a straightforward manner. First, the current resource and quality limits are compared to the limits that have been valid before. If they differ, according codes for ρ are returned. This indicates that resource requirements or the mining quality should be increased or decreased. After that, function φ is used to decide if resources are currently under- or overutilized, or will be in the next δ time units. This is done using the current parameter values and a filling factor f, e.g., if the resource requirements at time $t+\delta$ are less than $f = 70\%$ of $\mathcal{R}^{\top}$, then $\rho = 2$. The computation of φ for time $t + \delta$ is possible, because the size of the PTF is monotonically increasing with time. The computation based on $t+\delta$ ensures that all constraints will be met until the adaptivity component is triggered again at time $t + \delta$. It is not necessary to check if the quality remains within its limits, as Q_1 and Q_2 are constant for fixed parameter values, and thus quality limits are obeyed.

8. CONCLUSION AND OUTLOOK

In this paper, we presented a model to apply quality-driven resource adaptivity to stream-mining algorithms. First, we identified quality measures as well as parameters and observations of stream-mining algorithms. We then presented a correlation model that captures the correlations of parameters, observations, resource requirements, and quality measures. This model is used to formalize the objective functions of a multiobjective optimization problem, which is used to determine parameter values that maximize quality measures with respect to given resource and quality constraints. We presented the notion of Pareto-optimal solutions to the optimization problem, where each solution refers to parameter values that maximize at least one of the quality measures. Furthermore, we also presented three different approaches to determine a unique optimal solution. Finally, we discussed two approaches for schedules to invoke the resource adaptivity component. We illustrated the feasibility of the model on the basis of the CluStream algorithm.

Future work has do be done in applying the model to more algorithms and evaluating them in real-world scenarios, where the quality of the mining results can be constrained or judged by a domain expert. Further focus has to be put on the specific sub-tasks of the model, such as alternatives for solving the optimization problem, lightweight prediction approaches for proactive adaptivity, and the actual resource overhead of applying the model. Despite the fact that this introduces another small overhead, we believe that such an approach is mandatory to make stream mining meaningfully applicable in current stream-based application domains with evergrowing data volumes and system requirements.

9. REFERENCES

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB*, 2003.

[2] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin*, 26(1), 2003.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and D. Thomas. Operator scheduling in data stream systems. *VLDB Journal*, 13(4), 2004.

[4] H. Berthold, S. Schmidt, W. Lehner, and C.-J. Hamann. Integrated resource management for data stream systems. In *SAC*, 2005.

[5] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE'01*, pages 421–432, 2001.

[6] Y. Chi, H. Wang, and P. S. Yu. Loadstar: load shedding in data stream mining. In *VLDB*, 2005.

[7] C. A. C. Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1, 1999.

[8] M. Ehrgott. *Multicriteria Optimization*. Springer Berlin Heidelberg, 2005.

[9] C. Franke. *Adaptivity in Data Stream Mining*. PhD thesis, University of California at Davis, 2009.

[10] C. Franke, M. Hartung, M. Karnstedt, and K.-U. Sattler. Quality-aware mining of data streams. In *International Conference on Information Quality (ICIQ)*, 2005.

[11] A. A. Freitas. A critical review of multi-objective optimization in data mining: a position paper. *SIGKDD Explorations*, 6(2), 2004.

[12] M. M. Gaber and P. S. Yu. A holistic approach for resource-aware adaptive data stream mining. *Journal of New Generation Computing*, 25(1), 2007.

[13] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu. Mining frequent patterns in data streams at multiple time granularities. *Next Generation Data Mining*, 2003.

[14] A. Jain, E. Y. Chang, and Y.-F. Wang. Adaptive stream resource management using kalman filters. In *SIGMOD*, 2004.

[15] M. Karnstedt, C. Franke, and M. M. Gaber. A model for quality guaranteed resource-aware stream mining. In *5th International Workshop on Knowledge Discovery from Ubiquitous Data Streams*, 2007.

[16] D. Klan, K. Hose, M. Karnstedt, and K. Sattler. Power-Aware Data Analysis in Sensor Networks. In *ICDE Demonstrations Track*, 2010.

[17] D. Lee and W. Lee. Finding maximal frequent itemsets over online data streams adaptively. In *ICDM*, 2005.

[18] W.-G. Teng, M.-S. Chen, and P. S. Yu. Using wavelet-based resource-aware mining to explore temporal and support count granularities in data streams. In *SDM*, 2004.

[19] D. A. Van Veldhuizen and G. B. Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation*, 8(2), 2000.

[20] M. Vlachos, D. S. Turaga, and P. S. Yu. Resource adaptive periodicity estimation of streaming data. In *EDBT*, 2006.

[21] P. S. Yu. Data stream mining and resource adaptive computation. In *International Conference on Database Systems for Advanced Applications (DASFAA)*, 2005.