"What makes my queries slow?": Subgroup Discovery for SQL Workload Analysis

Youcef Remil, Anes Bendimerad, Romain Mathonat, Philippe Chaleat, Mehdi Kaytoue

”What makes my queries slow?”: Subgroup Discovery for SQL Workload Analysis

Youcef Remil[1,2,3], Anes Bendimerad[2], Romain Mathonat[2], Philippe Chaleat[2], Mehdi Kaytoue[1,2]

[1] Univ Lyon, INSA Lyon, CNRS, LIRIS UMR 5205, F-69621, Lyon, France
[2] Infologic, 99 avenue de Lyon, 26500 Bourg-Lès-Valence, France
[3] yre@infologic.fr

***Abstract*—Among daily tasks of database administrators (DBAs), the analysis of query workloads to identify schema issues and improving performances is crucial. Although DBAs can easily pinpoint queries repeatedly causing performance issues, it remains challenging to automatically identify subsets of queries that share some properties only (a pattern) and simultaneously foster some target measures, such as execution time. Patterns are defined on combinations of query clauses, environment variables, database alerts and metrics and help answer questions like *what makes SQL queries slow*? *What makes I/O communications high?* Automatically discovering these patterns in a huge search space and providing them as hypotheses for helping to localize issues and root-causes is important in the context of explainable AI. To tackle it, we introduce an original approach rooted on *Subgroup Discovery*. We show how to instantiate and develop this generic data-mining framework to identify potential causes of SQL workloads issues. We believe that such data-mining technique is not trivial to apply for DBAs. As such, we also provide a visualization tool for interactive knowledge discovery. We analyse a one week workload from hundreds of databases from our company, make both the dataset and source code available, and experimentally show that insightful hypotheses can be discovered.**

***Index Terms*—Database, Workload Analysis, Data Mining, Subgroup Discovery, Explainable AI, Data Visualisation**

I. Introduction

It is indisputable that data has become a crucial part of software and IT platforms. This makes the database and its management system a critical component. Thus, researchers and engineers have spent a significant effort to make the interaction with data as reliable and efficient as possible. A data-driven strategy based on query workload analysis has proven its efficiency to address a large variety of related problems. These methods automatically analyze the set of logs and queries run on the database to perform tasks such as index recommendation [1], [2], query recommendation [3], [4], anti-pattern detection [5]–[7], modeling user and application behavior [8], [9]. The usability of these data on such tasks strongly depends on their representation. That is why several methods have been proposed to transform the data into simplified forms before performing the main task, for example, workload compression [2], efficient parsing [10] or embedding [11] of SQL queries. Then, a myriad of Machine Learning methods have been evaluated on different workload analysis tasks. For example, clustering approaches have been exploited to delineate hot spots of user interests [12], to summarize workloads [13], to identify insider threats [14]. NLP techniques have been used to embed SQL queries in vector representations that are guided by the target application [11]. In [15], a neural network approach is proposed to help end-users and administrators compose SQL queries.

In this paper, we address a novel workload analysis problem that can uncover many kinds of tasks. We aim to design a method that efficiently brings answers to the generic question: *how to characterize SQL queries that foster some properties of interest?* A concrete example of such question is: *what makes queries slow?* Here the goal is to identify the characteristics and the context in which the execution time of queries is large. An example of results for such question is:

$$\text{Table} = X \wedge \text{Where attribute} = Y \longrightarrow \text{high execution time}$$

Answering this question can be extremely useful for performance optimization problems. Similarly, several other questions of this kind may occur: how to characterize queries that over-consume the I/O communication? In which context SQL queries significantly increase concurrency issues? etc. To address this problem, we propose a unified and powerful framework rooted in the Subgroup Discovery approach. Subgroup Discovery [16], [17] is a data mining task that aims, among many other possibilities, to identify patterns describing parts in a dataset where the distribution of the target variable significantly deviates from the “norm”, i.e. from its distribution in the whole dataset. Typically, the discovered subgroups are easily interpreted by the experts. Coming back to our previous example, the discovered subgroup consists of all the queries that verify the constraint “Table = X $\wedge$ Where attribute = Y”. This subgroup is interesting because its average execution time is significantly greater than expected.

Subgroup Discovery has proven its efficiency in different fields such as physics [18], education [19] and neuro-science [20]. To the best of our knowledge, our work is the first to exploit this approach to address a generic workload analysis problem. The efficiency of such method is challenging as it strongly depends on several complex criteria: (1) the data needs to be introduced to the algorithm in the right format, (2) a relevant pattern language needs to be defined (the language used to select subsets of queries), (3) we need to choose the

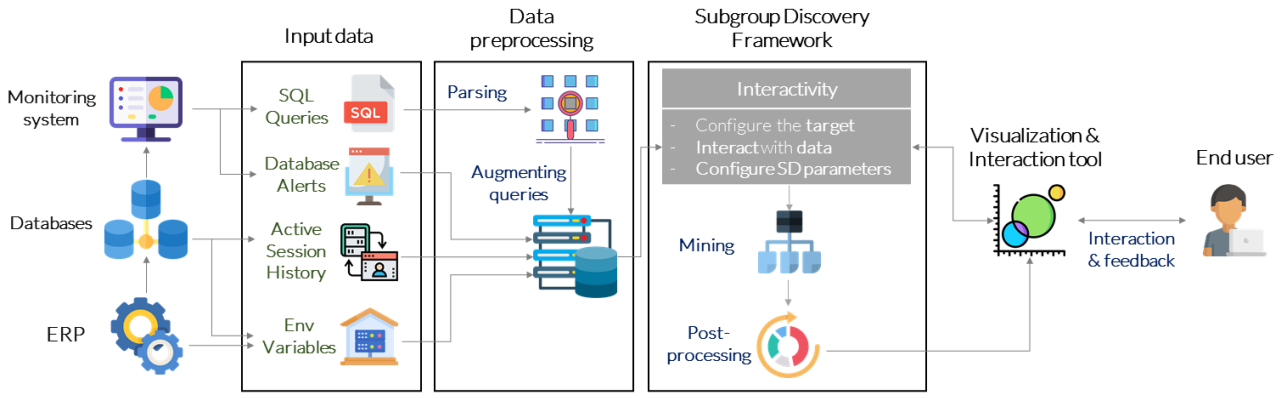


Figure 1: Overview of our Subgroup Discovery framework for SQL workload analysis.

right function to measure the interestingness of a subgroup w.r.t. the target problem, (4) resulting subgroups need to be interpretable and their interactive mining needs to be enabled.

Contributions. We introduce an efficient Subgroup Discovery framework that meets all the aforementioned criteria for Workload Analysis. We first propose a data pre-processing step to prepare the SQL workload. We parse queries to extract important attributes (e.g., tables, fields, operations). We have extended the Mozilla parser [21] with new features to extract all the information we need from the queries* (e.g., handling alias and nested queries). Moreover, we augment queries with other relevant information: execution time, performance metrics of the DBMS, environment variables of the system, and anomaly alerts guided by expert knowledge. Then, we define a suitable pattern language, and integrate a diverse set of interestingness measures whose choice can be directed by the target application. We provide exact and heuristic algorithms to identify subgroups of interest. Furthermore, we integrate a visual tool that enables the user to interact with the algorithm, and iteratively learn from the provided results. The whole process of the proposed approach is summarized in Figure 1.

Outline. The remainder of the paper is organized as follows. Sec II presents the raw data, the pre-processing strategy, as well as an informal description of the studied problem. Sec III formally defines the problem settings, introduces the interestingness measures in Sec III-B and algorithms in Sec III-C and Sec III-D. Then, Sec III-E presents our interactive visual tool that enables the user to easily annotate data and design the target task. A thorough empirical study is detailed in Sec IV to quantitatively and qualitatively evaluate the proposed approach. Sec V presents related work before we conclude and present future directions.

II. Methodology

We conduct our analysis on a workload W of 150K unparametrized SQL statements gathered from more than 400 databases supervised in our company, sharing almost the same database schema. We efficiently parse queries to extract tables and attributes for each type of SQL clause. Queries are then augmented with several database metrics and supervision alerts, resulting in thousands of properties helping in contextualizing the subgroup discovery.

A. Raw Data

SQL queries. We define the workload as a set $W = \{q_1, ..., q_n\}$ where each query $q \in W$ is a `SELECT` statement that contains (1) the SQL text q_{text}, (2) the query execution time q_{time} and (3) the number of rows returned q_{nrows}.

Environment features. Each database is queried by an application, an Enterprise Resource Planning software (ERP) that we develop in the company. As such, the application identifier (*serverName)* and its major and minor versions are considered (*softwareVersion*, *codeVersion*). The database properties are its vendor/version (*dbVersion)*, its schema among 6 main families (*declination*), the size of the database server memory (*dbMemory*), the maximum database memory usage (*sgaMax*), the maximum number of processes (*dbProcesses*), the minimum and the maximum size of pool (*jdbcMin*, *jdbcMax*), the limit on the number of cursors per database session (*dbCursorMax*).

Active Session History. Active Session History (ASH) [22] was introduced in Oracle 10g, and then in other database systems such as PostgreSQL [23]. An active session is a database session waiting for some resource such as CPU, System I/O or Network. ASH provides the number of sessions waiting for each category of resource, per interval of time. It gives a temporal distribution that can be of high interest for diagnostics and tuning. For example, one may use this data to identify queries that unexpectedly over-consume network as they generate many network waiting sessions.

Alerts. Our monitoring system triggers rule-based alerts when anomalies are observed on the database environment. For this analysis, we considered 4 alerts: (1) when the number of active sessions is unusually large (*manyActiveSessions*), (2) when some sessions remain blocked during a significant time (*blockedSessions*), (3) when the size of the pool is close to

*Code and datasets available on https://github.com/RemilYoucef/sd-4sql

Table I: Dataset features.

Query properties	Environment variables		Alerts	Oracle ASH	
query	serverName	dbMemory	manyActiveSessions	application	concurrence
day	declination	sgaMax	blockedSessions	configuration	network
hour	softwareVersion	dbProcesses	poolAlmostFull	administrative	cpu
time	codeVersion	jdbcMin	anomalyASH	systemI/O	userI/O
nrows	dbVersion	jdbcMax		queuing	scheduler
		dbCursorsMax		commit	

its maximum limit (*poolAlmostFull*), and (4) when there is an anomaly in the distribution of ASH (*anomalyASH*), e.g., an increase in the proportion of sessions waiting for network or systemI/O. We have augmented the queries with the alerts that co-occur with their execution. Each alert has four levels: *Info*, *Alarm*, *Critical* and *blocking*.

These features have been chosen with our DBAs. Our methodology is totally flexible and any numerical and categorical property can be considered as well. As related in Section V, it should be noticed that no prior work has invested such a combination of high dimensional features along with a very expressive representation of SQL queries that we present now.

B. Query transformation

A common preprocessing is to decompose, parse and tokenize SQL queries q_{text} to form a numerical vector where dimensions count the usage of data tables and attributes [3], [4], [24]. We have used the readily available Mozilla parser [21] that provides an SQL syntactic tree in XML that we then parse. We normalize the case sensitivity and remove irrelevant terms such as constants and logical operators. Tokens are then associated with the clauses they belong to, by adding to the token-name a prefix that indicates the clause in which each token appears. For instance in Figure 2, the table `model` appears in the `FROM` clause of the query, so its token will be `FROM_model`. For each token, we provide the number of time it appears in each SQL clause.

Then, we extended the Mozilla parser in two ways. Firstly we needed to consider not only SQL queries but also Hibernate queries used in our ERP as the ORM layer: we added the reserved keywords for hibernate queries such as `JOIN FETCH` in the original parser. Second, and most importantly, unlike several existing parsers [3], [4], [24], our parser can handle nested queries while preserving all the structure of the query. Furthermore, substitutes for temporary table names known as *aliases* are not removed as also done in many parsers [3], [25]. Aliases are rather used to figure out for example, to which table belongs a column in a `SELECT` clause or a predicate in `WHERE` or `GROUP BY` clause. The interest in keeping and using alias appears when the queries are nested or the query contains a join clause, or involves several tables. In this way, as the example in Figure 2 shows, if two columns of different tables have the same name, they will be encoded differently unlike the approach proposed by [25]. Inspired by the work of [26], we also consider the function calls present in the SQL statement, as an independent clause. The source code is available as mentionned in the introduction.

Raw SQL query

```
SELECT m.ik
FROM model AS m
JOIN prod AS p
WHERE m.ik = p.ik
  AND m.uex = p1
  AND (m.uex in collection0
      OR m.ik in collection1)
  AND (m.dossierinfo = p3
GROUP BY m.ik
HAVING (COUNT(DISTINCT p.ik) = p2)
AND (SUM(m.nbembal) = MAX (p.nbembal))
```

Our parsing result	**Parsing result of** [25]
SELECT_model.ik ⟶ 1	SELECT_ik ⟶ 1
FROM_model ⟶ 1 JOIN_prod ⟶ 1	FROM_model ⟶ 1 FROM_prod ⟶ 1
WHERE_model.ik ⟶ 3 WHERE_model.uex ⟶ 1 WHERE_model.dossierinfo ⟶ 1 WHERE_prod.ik ⟶ 1	WHERE_ik ⟶ 4 WHERE_uex ⟶ 1 WHERE_dossierinfo ⟶ 1
GROUPBY_model.ik ⟶ 1	GROUPBY_ik ⟶ 1
HAVING_prod.ik ⟶ 1 HAVING_model.nbembal ⟶ 1 HAVING_prod.nbembal ⟶ 1	HAVING_ik ⟶ 1 HAVING_nbembal ⟶ 2
COUNT_prod.ik ⟶ 1 SUM_model.nbembal ⟶ 1 MAX_prod.nbembal ⟶ 1	

Figure 2: Example of parsing an SQL query.

Finally, it is noteworthy that we do not group semantically equivalent queries under a canonical form as done in [10]: the way a query is written can impact its execution plan, thus, execution time.

C. Data Model

We unify the different data sources into a dataset defined by a pair (O, A), where $O = \{o_i\}_{1 \leq i \leq n}$ is a set of objects that refer to the queries, and $A = (a_j)_{1 \leq j \leq m}$ is a vector of attributes. Each attribute $a : O \longrightarrow dom(a)$ is a function that maps queries to values in its domain $dom(a)$. Consequently, $a(o)$ denotes the value of the attribute a for the object o. $dom(a)$ is given by $\mathbb{R}$ if a is *numerical*, by a finite set of categories C_i if a is *nominal (categorical)*, or by $\{0, 1\}$ if a is *Boolean*. A nominal attributes with a total ordering of its values is called an *ordinal nominal* attribute. These notations are illustrated in Table II with a dataset of 11 objects $O = \{o_1, ..., o_{11}\}$ referring to queries described by 12 attributes. *Server name* is nominal and has two possible values: *LYN* and *BLV*. *manyActiveSessions*, referring to an alert, is an

O	FROM		WHERE			ENV features		Alerts	ASH	q_{nrows}	q_{time}	
	a_1 Verrou	a_2 Cumulof	a_3 Verrou.ik	a_4 Verrou.date	a_5 Cumulof.ik	a_6 Soft version	a_7 Server name	a_8 manyActiveSessions	a_9 Concurrency	a_{10} nrows	a_{11} time	a_{12} slow
o_1	1	0	1	0	0	v2	LYN	Alarm	22	10	2.15	0
o_2	1	0	1	1	0	v1	BLV	Critical	3	1	15.81	1
o_3	0	1	0	0	1	v1	BLV	Critical	15	27	1.14	0
o_4	1	1	0	1	1	v2	LYN	Alarm	31	12	10.87	1
o_5	1	1	1	0	1	v3	LYN	Alarm	11	25	2.1	0
o_6	1	0	1	2	0	v3	LYN	Critical	6	100	17.93	1
o_7	1	1	1	1	1	v2	LYN	Info	27	1	15.8	1
o_8	0	1	0	0	1	v2	BLV	Alarm	9	37	9.95	0
o_9	1	0	1	0	0	v3	BLV	Critical	10	112	8.95	0
o_{10}	0	1	0	0	1	v2	BLV	Alarm	7	1	14.7	1
o_{11}	0	1	0	0	0	v2	LYN	Info	25	16	1.0	0

Table II: Toy Example of a dataset (O, A).

ordinal attribute with 3 levels {*Info*, *Alarm*, *Critical*}. *time* is numerical and gives the execution time that a query takes. *slow* is binary, it equals 1 when the query time exceeds 10 seconds, 0 otherwise. As mentioned in Sec II-B, when parsing a query, we keep the count of each token associated with each clause, thus, each token is a numerical attribute $a_j \in A$ such that : $dom(a_j) = \mathbb{N} \subset \mathbb{R}$. For example, `Verrou.data` is numerical and represents the number of times this attribute appears in the `WHERE` clause for each query, e.g., $a_4(o_6) = 2$ means that for the 6-th query in the dataset the attribute `Verrou.date` appears twice in the `WHERE` clause.

D. Characterizing Discriminant Queries

The goal of subgroup discovery is to find subsets of objects that are statistically *the most interesting* with respect to a property of interest i.e., the *target*. For example, we seek to characterize queries that have large execution times, i.e., *slow* queries. Thus, the *target concept* can be the binary attribute *slow*, and we will identify interpretable descriptions of subgroups that maximize the proportion of queries having $slow = 1$. In Table II, a discriminant subgroup can be defined by queries that include the attribute `WHERE_Verrou.date`. Indeed, 100% of these queries are *slow*, while only 45% of overall queries are *slow*. We refer to this subgroup by its description : "`WHERE_Verrou.date` $>$ 0". Another interesting example consists in queries that correspond to the following description: "`WHERE_Cumulof.ik` $= 1 \wedge$ `Soft. version` = v2", with a proportion of 75% of *slow* queries. Given a large number of attributes, we end up with a very huge set of possible conjunctive combinations. Therefore, it becomes challenging to identify those descriptions that are the most significantly discriminant. This is where an automatic Subgroup Discovery approach can be extremely helpful. Such approach usually identifies interesting results by performing a deep search through the set of candidates hypotheses and scores each of them with a function that assesses their interestingness. In addition to this purely automatic approach, human expertise can be useful in guiding the search, since subgroup discovery involves an iterative and interactive process. The first subgroup ("`WHERE_Verrou.date` $>$ 0") can inform experts that an index is probably missing on the attribute `Verrou.date`. The previous examples use a binary attribute (*slow*) as the target concept, but subgroup discovery can also employ numerical targets or even complex models over multiple targets. In the following, consider that the target concept is the numerical attribute *time*. The subgroup defined as "`FROM_Verrou` $> 0 \wedge$ manyActiveSessions = *Critical*" is statistically interesting, because its average *time* of 14.22s is relatively large compared to the average over the whole dataset estimated by 9.12s. The last example would not have been impressive if we opted for the binary target *slow*, because this subgroup is characterized by 3 queries, one of which is barely less than 10s. It is also noteworthy that the size of subgroups is often taken into account to assess their quality. In fact, we are generally interested by discriminant subgroups that cover a large number of queries, as statistically more significant.

III. Discriminant Pattern Discovery in Workloads

A. Introduction to Subgroup Discovery

Descriptive attributes and target. One needs to specify a target attribute $t \in A$ that is suitable for the target application. For example, if we want to characterize slow queries, the target attribute t will be the execution time (q_{time}). In this paper, we consider both cases where t can be Boolean or numerical. Another important question is: which attributes do we want to use to characterize interesting subgroups? These are called descriptive attributes and denoted $A_D \subseteq A \setminus \{t\}$, and $|A_D| = m_D$. A pattern language $\mathcal{D}$ is then defined over descriptive attributes. A pattern $d \in \mathcal{D}$ is a constrained selector of subset of objects using their descriptive attribute values A_D. More precisely, the pattern language is defined as $\mathcal{D} = \bigtimes_{i=1}^{m_D} \mathcal{D}_i$ where $\mathcal{D}_i$ is a selector defined over $a_i \in A_D$, and given by the set of all possible intervals in $\mathbb{R}$ if a_i is numerical, the set $\{C_i, \emptyset\} \cup \{\{c\} \mid c \in C_i\}$ if a_i is categorical, or $\{\{0,1\},\{0\},\{1\}\}$ if a_i is Boolean. A pattern $d \in \mathcal{D}$ is then given by a set of restrictions over each descriptive attribute (i.e. $d = (d_i)_{1 \leq i \leq m_D}$).

Linking patterns and objects. A pattern $d = (d_i)_{1 \leq i \leq m_D}$ is said to *cover* an object $o \in O$ iff $\forall a_i \in A_D : a_i(o) \in d_i$. The set of all objects covered by a pattern d is called the extent of d and denoted $ext(d) = \{o \in O \mid d \text{ covers } o\}$. In Table II, consider the case where we have 3 descriptive attributes $A_D = \{$`FROM_Verrou`, `FROM_Cumulof`, `Server name`$\}$. An example of pattern is $d = ($`FROM_Verrou` $\in \mathbb{N}$,

FROM_Cumulof ≥ 1, Server name = LYN). This pattern covers objects in which FROM_Cumulof appears at least once in the query and the Server name is LYN. These objects are $ext(d) = \{o_4, o_5, o_7, o_{11}\}$.

Subgroup definition. A subgroup is any subset of objects $s \subseteq O$ that can be selected using a pattern d over descriptive attributes A_D. The set of all possible subgroups is denoted $\mathcal{S} = ext(\mathcal{D}) = \{ext(d) \mid d \in \mathcal{D}\}$. In other terms, a subgroup is a set of objects that can be characterized with some restrictions of attributes, turning it interpretable to the user.

Subgroup interestingness. A measure $\phi : \mathcal{S} \to \mathbb{R}; s \mapsto \phi(s)$ is a mapping that evaluates the quality of a subgroup s w.r.t. the property of interest. The greater is $\phi(s)$, the more interesting is s. The choice of ϕ depends on the target application. In Sec III-B, we define several relevant measures that we have exploited to analyze SQL queries, and we explain how to choose the right measure according to the end user goal.

Problem statement. Given a user specified parameter k, find the top-k subgroups with the highest values of the interestingness measure ϕ. Formally, find the subgroup set:

$$\mathcal{R} = \{s \in \mathcal{S} \mid rank(s) \leq k\},$$

where $rank(s)$ gives the rank of s w.r.t. its score ϕ, that is: $rank(s) = |\{s' \in \mathcal{S} \mid \phi(s') > \phi(s)\}| + 1$.

B. Measuring subgroup interestingness

We present measures that can be used as ϕ to assess the quality of subgroups. It is generally agreed that interesting discriminant subgroups are those that maximize the deviation of the target attribute t and whose size $|s|$ is sufficiently large. In fact, one prefers discriminant subgroups that have large sizes as they are deemed more significant, i.e., there existence in the dataset is less probable to be due to chance. Many of existing measures belong to the popular family of Klösgen functions [16] defined given the parameter $a \in [0, 1]$:

$$\texttt{Klösgen}_a(s) = sup(s)^a \cdot (\mu(s) - \mu(O)),$$

where the support $sup(s) = \frac{|s|}{|O|}$ measures the proportion of objects from O that belongs to s, and the target mean $\mu(s) = \sum_{o \in s} \frac{t(o)}{|S|}$ gives the average value of the target attribute t in s. Thus, the higher is $sup(s)$, the higher is $\texttt{Klösgen}_a(s)$. But also, $\texttt{Klösgen}_a(s)$ is maximized when the deviation of $\mu(s)$ regarding its overall mean $\mu(O)$ is maximized. The choice of a affects the importance of $sup(s)$ on the final value of interestingness, and conducts to measures with different statistical interpretations. These measures are presented in what follows.

Average function u. Also called unusualness [27], it is the Klösgen function with $a = 0$, that is: $u(s) = \mu(s) - \mu(O)$. It can be used when we do not want to impact the score of subgroups by $sup(s)$. When used, this measure is generally combined with a threshold constraint on a minimum size of returned subgroups, to avoid retrieving very small ones. $u(s)$ provides a subgroup ordering that is identical to another popular measure: $\texttt{Lift}(s) = \frac{\mu(s)}{\mu(O)}$.

WRAcc measure [28]. It is one of the most popular measures in SD. It corresponds to the Klösgen function with $a = 1$:

$$\texttt{WRAcc}(s) = sup(s) \cdot (\mu(s) - \mu(O)).$$

For the specific case when t is binary, it can be written as:

$$\texttt{WRAcc}(s) = \texttt{Pr}(o \in s \wedge t(o) = 1) - \texttt{Pr}(o \in s) \cdot \texttt{Pr}(t(o) = 1),$$

where Pr is the probability of an event to happen. Theoretically, the more s is statistically dependent of true target values ($t(o) = 1$), the higher is $|WRAcc(s)|$.

Mean-test. A drawback of WRAcc is that, in many tasks, it over-scores subgroups with large support despite their limited unusualness. For this reason, many methods have preferred to use the Mean-test, which is the Klösgen function with $a = 0.5$:

$$\texttt{Mean-test}(s) = \sqrt{sup(s)} \cdot (\mu(s) - \mu(O)).$$

From a statistical point of view, it was proven that this measure provides an equivalent ordering than the Binomial test [29].

T-score. A limitation of Klösgen functions is that they do not optimize the dispersion of the target attribute in subgroups. This could lead to inconsistent statements, particularly when the dataset contains many outliers. In fact, $\mu(s)$ is sometimes not representative of target values in the subgroup s, if it contains few outliers with extreme values of t. To address this issue, one of the measures that incorporate the cohesion of the subgroup is the T-score [30], defined as:

$$\texttt{T-score}(s) = \frac{\sqrt{sup(s)}}{\sigma(s)} \cdot (\mu(s) - \mu(O)),$$

where $\sigma(s)$ is the standard-deviation of target values t in the subgroup s. The smaller is $\sigma(s)$, the more cohesive are values of t in s, and thus the higher is $\texttt{T-score}(s)$. This measure reflects the significance of the deviation of target values in a subgroup using a Student's t-test. However, one should avoid a direct statistical interpretation of the T-score if the target attribute is not normally distributed and the subgroup size is small, e.g., $|s| < 30$.

Median-based measures q_med. Another way to reduce the impact of outliers on subgroup scores is to estimates values of t in s using its median $\texttt{med}(s)$ instead of its average $\mu(s)$ in the Klösgen function, as the median estimator is more robust to noise [31]:

$$\texttt{q\_med}(s) = sup(s)^a \cdot (\texttt{med}(s) - \texttt{med}(O)).$$

C. Algorithms

Once the pattern language is defined and the subgroup interestingness is chosen, it remains to explore the search space in order to identify the top-k subgroups. Computational complexity of SD problem is known to be prohibitive due to the huge size of the search space $|\mathcal{S}|$ that increases exponentially w.r.t. $|A_D|$. Many algorithms have been proposed to efficiently traverse the search space. Some of them provide exact results, others are heuristic but scale better. [32] proposed a Python

implementation of the most popular SD algorithms. Since it does not support all the relevant measures for our case study, we have extended this framework to incorporate: the support, the `T-score`, as well as median-based measures `q_med`. We exploit two methods: (1) an exact algorithm based on a depth-first search, (2) a heuristic algorithm that uses beam-search.

Depth-first algorithm. This approach exhaustively explores the search space $\mathcal{S}$ in a depth-first manner. After defining an order relation between patterns, the search space forms a lattice structure with the empty pattern as a supremum and the pattern containing all the selectors as infimum. Then, this lattice is explored in depth. We start from the empty pattern $d = (a_i \in dom(a_i) | a_i \in A_D)$, i.e., no restriction for any attribute. Then, a refinement operator is recursively applied on selectors of d, continuously making it more restrictive. Refinements can be operated by adding a symbolic attribute value, or adding a numerical attribute cut point. As the search space can be extremely large, a naive enumeration of subgroups fails. For this reason, the exact algorithm uses many techniques to optimize the exploration. Some anti-monotonic constraints are generally used, such as a minimum support δ, i.e., if a pattern covers less than δ objects then this pattern is not refined anymore, as its refinement necessarily covers less than δ objects. Furthermore, tight optimistic estimates `TOE` [31] are used. These functions allow to efficiently upper bound all the subgroup interestingness values in a whole branch of the search space. If the `TOE` of a branch is lower than the score of the top-k already found subgroup, then the branch is pruned, as it does not contain any subgroup with a score higher than the already found top-k. Other optimization strategies are used as well. to [31] for further details.

Beam-search algorithm. Heuristic methods are deemed useful in many scenarios when the number of descriptive attributes is large, and thus exact methods become slow or infeasible. They try to find as good patterns as possible in a short time by evaluating only promising candidates. The most popular heuristic approach is Beam-search [33]. This approach performs a heuristic level-wise search over the pattern lattice. It requires to specify the width parameter $w \in \mathbb{N}$, which is the maximum number of patterns kept in each level of the lattice. It starts from the empty pattern $d = (a_i \in dom(a_i) | a_i \in A_D)$. Then, it recursively goes to the next level by refining patterns of the current level and selecting the top-w refined patterns that maximize the interestingness. These top-w patterns are then refined again to continue to a deeper level. At the end, the algorithm selects the top-k subgroups among all the top-w ones selected from each level.

D. Reducing information redundancy

The process of selecting interesting subgroups considering only the discriminative measure may result in strongly overlapping subgroups, that are distinct patterns covering almost the same objects. For instance in Table II, the two different patterns : $d_1 = (\texttt{FROM\_Verrou} \geq 1)$ and $d_2 = (\texttt{WHERE\_Verrou.ik} \geq 1)$ are highly correlated as they cover more or less the same objects. In order to provide interesting but diverse patterns, and to reduce information redundancy in the subgroup set $\mathcal{R}$, we propose two different solutions based on the *Jaccard similarity* [34]. This metric measures the similarity of two subgroups patterns as a fraction between the intersection and the union of their extents. For the two example patterns d_1 and d_2:

$$sim(d_1, d_2) = J(d_1, d_2) = \frac{|ext(d_1) \cap ext(d_2)|}{|ext(d_1) \cup ext(d_2)|} = \frac{6}{7}$$

Greedy Selection. The greedy approach constructs iteratively the non redundant subgroup set $\mathcal{R}'$. In each iteration, the best subgroup $s^\star$ in the initial subgroup set $\mathcal{R}$ is identified, and added to $\mathcal{R}'$. Afterwards, we remove from $\mathcal{R}$ all the subgroups whose similarity with $s^\star$ exceeds a specified threshold. This process is repeated until $\mathcal{R}$ becomes empty. However, this technique requires the user to specify an appropriate threshold for the *Jaccard similarity*. For a large enough threshold, one can still end up with overlapping subgroups. On the other side, a small threshold can lead to the suppression of interesting subgroups. Thus, this method is sensitive to the threshold which must be chosen empirically.

Hierarchical Clustering. To get a more complete and understandable overview of the resulting subgroups, agglomerative hierarchical clustering [35] is performed on the result set $\mathcal{R}$. At the bottom of the hierarchy, each subgroup forms a singleton cluster, and pairs of clusters are then merged as one moves up the hierarchy. This clustering is computed using the *Jaccard dissimilarity* defined as $1 - $*Jaccard similarity*. As a result, the hierarchical clustering produces a binary clustering tree or a *dendrogram*. It represents a hierarchy of partitions, hence, it is possible to choose one partition by truncating the tree at a given level. Unlike the greedy approach, the user can specify how many non-redundant subgroup patterns she wants without having to specify a dissimilarity threshold.

E. Interactive SD for Workload Analysis

In practice, an effective SD approach needs to be iterative and interactive, to make it possible to incorporate subjective criteria as well as human expertise. Indeed, the interestingness of subgroups strongly depends on the end user preferences and her prior knowledge about the data. This interactive process should efficiently allow the user to explore the region of her hypothesis space, and possibly improve the quality of the extracted pattern. In that process, the user sets the parameters of the approach including the measure and the algorithm to get a visualisation of the retrieved patterns. Afterwards, the user proceeds to the validation of these patterns and checks for the quality of the provided knowledge, while guiding the post processing phase to refine relevant patterns. Several subgroup discovery algorithms have been embedded into software tools such as KEPLER [36], SubgroupMiner [37] and VIKAMINE [38] that provide a graphical interface allowing the user to choose viewing options and select appropriate parameters for her task. Different from those just mentioned, the graphical tool we provide enables the user to: (1) use visual filters to

select the subset of the data she wants to mine with SD, and (2) visually constitute a binary *target concept* that she aims to discriminate, by flexibly selecting data inside widgets such as scatter plots. Moreover, the tool allows for setting the desired parameters of the task and the visualization of the extracted patterns along with their associated statistics. Figure 4 shows the main window of this tool, which is described in Sec IV-C.

IV. Evaluation

We report the experimental study that we conducted to evaluate the efficiency of our Subgroup Discovery approach. First, we validate that the proposed framework is able to characterize discriminant subgroups that are *statistically* the most interesting w.r.t. different target problems. Then, we report through a quantitative analysis the execution time for each algorithm with different parameters including the number of patterns k and rules depth. Finally, we present the different features provided by our visualization tool. Further study of measures and post-processing are differed in the supplementary materials due to lack of space.

Experiments Setup. Experiments are conducted on an SQL workload that contains hibernate queries run on our production-environment servers for a period of one week. Since the effective number of queries is extremely large, our monitoring system records only those whose execution time exceeds 5 seconds. We augment these queries with other relevant information described in Table I, as explained in Sec II-A. This dataset contains 148,796 queries described by 8,691 features. Table III displays the characteristics of the overall dataset. Note that the dataset is extremely sparse, with only 0.45% of non-zero values. The proposed framework extended the library Pysubgroup [32] to support more relevant measures, specifically for numerical targets (e.g., median-based measure). All the experiments presented were run on a single machine with (Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz with 32GB RAM).

A. Qualitative Analysis

Use cases. To show the ability of the proposed framework to perform several tasks with different goals, we use it to address the following diverse set of research questions:

- **RQ1**: What makes queries very slow?
- **RQ2**: In which context do queries present concurrency issues?
- **RQ3**: How to characterize queries that co-occur the most with alerts of type *blocked sessions*.

Methodology. To properly conduct these experiments, we assess each use case in a specific context according to the industrial needs that we are facing. More precisely, each use case is carried out on a subset of data as follows: We evaluate **RQ1** on **D1**: queries that were executed on all sales servers with at least 100 users. Note that the sales declination constitutes 74.04% of the data. **D2**: queries that invoke the `MVTREALISE` table, knowing that it is the most commonly queried table. **RQ2** is evaluated particularly on **D3**: the software version (V15_2), since it presents exclusively many concurrency issues, compared to other different software versions. For this purpose, we consider a binary target which refers to a potential issue if there is at least an average of 5 concurrent processes over a period of 10 seconds during the execution of the query. Finally, we evaluate **RQ3** on **D4**: a specific set of servers on which we observe an abnormal raising of blocked session alerts. The characteristics of the 4 sub-datasets are provided in Table III. For each of the studied scenarios, we choose the top-10 subgroups w.r.t. to the most adequate interestingness measure for the problem. The choice of the measure is made empirically by comparing the subgroups identified with each measure. We take the one that provides relevant but also interpretable patterns by performing a statistical distribution analysis and referring to human expertise. The resulting patterns are then processed to provide diverse and non-redundant ones, as described in Sec III-D. Results are given in Table IV where for each subgroup pattern, we show its support, as well as its deviant quality compared to the dataset. For binary target problems, we compare the precision of each subgroup s given by : $prec(s) = \frac{|o \in s | t(o)=1|}{|s|}$ with the precision of the considered dataset.

Results and Analysis. Actionable and relevant subgroups have been identified in the different use-cases. In the dataset **D1**, we were interested in subgroups whose median execution time is significantly higher than the median of the dataset, while taking into account the subgroup size. We have chosen the measure `q_med` instead of the `Mean-test`, because we observed in particular for this example, that the mean is more sensitive to outliers. The subgroup (s_1) which covers all queries that involve the attribute `auditinfo.etat` in the `WHERE` clause has a very large median compared to the dataset, but only few objects. In Figure 3a, we show that its density distribution is too divergent and does not follow the usual distribution of original data. On the other hand, while the subgroup (s_2) includes all the 451 queries executed on the `cumulmultiple` table characterized by a large median, the subgroups (s_3) and (s_4) are subsets of (s_2) as they cover only its queries having the attributes `valzvcliX` and `valzvartX` respectively in their `WHERE` clause. As shown In Figure 3a, the deviation of (s_3) and (s_4) from the overall distribution is stronger than the deviation of (s_2), since they do not cover some slow queries present in (s_2). To better understand this result, we have examined the `cumulmultiple` table by highlighting the distributions of its attributes in Figure 3b. We then confirm that mostly the attributes `valzvcliX` and `valzvartX` cause the s_2 to be identified. In **D2**, we discretize the attribute time so that queries with an execution time higher than 10 seconds are considered as slow queries. For each measure, we obtain interesting results that incorporate the extended features (e.g., alerts in (s_9) and environment variables in (s_7)). For example, we found that all the queries on the `mvtrealise` table that

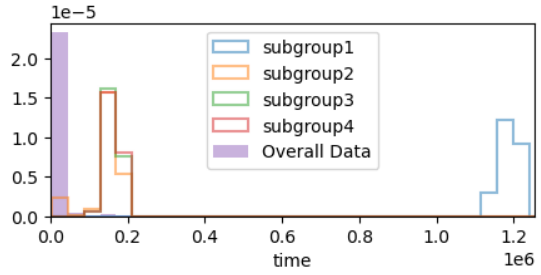
Table III: Datasets statistics.

Dataset	Queries	Features	FROM Tables	JOIN Tables	Projections	WHERE atts	HAVING atts	GROUPBY atts	ORDERBY atts	Sparsity
All	148796	8691	497	526	3740	3294	10	199	391	99.55%
D1	37149	4596	275	270	2036	1680	10	96	196	99.22%
D2	48823	246	1	1	86	85	2	21	11	84.27%
D3	3031	570	58	30	158	275	3	6	15	94.77%
D4	26735	3723	218	234	1658	1324	10	91	154	98.97%

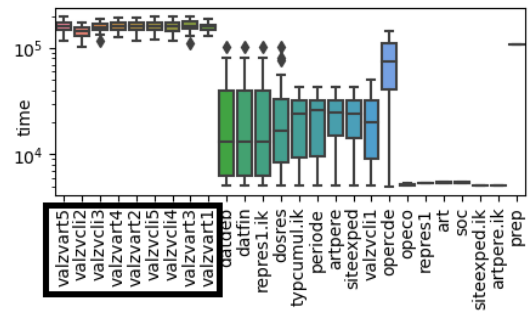
Table IV: Subgroup Discovery Results.

ID	Target	Measure	Subgroup patterns	Size	Quality
D1	**time** (Numerical)	Median	(s_1) : WHERE_stocks.gestion.modele.lot.prod.ref.auditinfo.etat > 0	8	$161 \times med\_dataset$
			(s_2) : FROM_ventes.cumuls.modele.cumulmultiple > 0	451	$21 \times med\_dataset$
			(s_3) : WHERE_ventes.cumuls.modele.cumulmultiple.valzvcli**X** > 0	45	$21 \times med\_dataset$
			(s_4) : WHERE_.ventes.cumuls.modele.cumulmultiple.valzvart**X** > 0	45	$21 \times med\_dataset$
D2	**slow** (Binary) $prec \simeq 60\%$	Lift	(s_5) : GROUPBY_stocks.gestion.modele.mvtrealise.refexterne > 0	131	$prec = 100\%$
			(s_6) : serverName = ServerX $\wedge$ systemI/O > 50	38	$prec = 100\%$
		WRAcc	(s_7) : WHERE_stocks.gestion.modele.mvtrealise.etatsynchro $> 0 \wedge$ jdbcMax < 200	20668	$prec \simeq 99\%$
			(s_8) : WHERE_stocks.gestion.modele.mvtrealise.auditinfo.datcre $> 0 \wedge$ dbVersion $= 2.3$	20675	$prec \simeq 99\%$
			(s_9) : manyActiveSessions = Alarm	44	$prec \simeq 93\%$
D3	**concurrence** (Binary) $prec \simeq 6\%$	Lift	(s_{10}) : FROM_.stocks.fichierbase.modele.produit $> 0 \wedge$ administrative $= 0.3$	8	$prec = 100\%$
		Binomial	(s_{11}) : serverName = ServerY $\wedge$ commit $> 0.7 \wedge$ systemI/O > 10.2	51	$prec \simeq 94\%$
D4	**blockedSess** (Binary) $prec \simeq 4\%$	Lift	(s_{12}) : JOIN_.commandesfactures.modele.histcdeligliv.applibudrist > 0	7	$prec = 100\%$
			(s_{13}) : WHERE_ventes.commandesfactures.modele.cdeligliv.bonliv.datdepart > 0	9	$prec \simeq 90\%$
		Binomial	(s_{14}) : anomalyASH = Critical	151	$prec \simeq 85\%$
			(s_{15}) : poolAlmostFull = Info	124	$prec \simeq 99\%$

are executed on ServerX[†] when the systemI/O is at least 50, last more than 10 seconds. Moreover, each time, the refexterne attribute is requested by the GROUP BY clause, the query takes very long time to execute. Unlike the lift measure which relies only on the precision of the subgroup, WRAcc takes the subgroup size into account. The subgroups (s_7) and (s_8) are very exceptional because they cover more than 42% of the queries while having an approximate precision of 99% compared to an overall precision of 60% i.e., these subgroups contain more than 70% of the slow queries. In **D3**, we aim to figure out the context in which queries encounter concurrency problems. The best results are achieved using the lift and binomial measures. Although the precision on the considered dataset is estimated to be only 6%, we extracted subgroups with a precision that exceeds 94%. The subgroup (s_{11}) alone constitutes 28% of objects that characterize a concurrency issue. Finally, for **D4**, we want to extract relevant hypotheses that reveal the context in which the *blocked sessions* alert is raised with blocking or critical level. This is generally due to the execution of a query which blocks a critical resource and puts new sessions on hold. We found that every time the table applibudrist is *joined* with another table, the alert is triggered. Another possible reason may be a process that queries a table with missing indexes. This is where the subgroup (s_{13}) allows quickly to check if this assumption is true on the datdepart attribute. We also observed that the concerned alert is highly correlated to both the two alerts described by the subgroups (s_{14}) and (s_{15}). It is worth noting that the proposed SQL parser has been effectively useful, since it helped to contextualize interesting subgroups of queries. In fact, we notice through the experiments that the extracted subgroup patterns include different SQL clauses (e.g., WHERE, GROUP BY, etc.).



(a) Subgroups distribution w.r.t time on D1 compared to overall data.



(b) Distribution of attributes in the table cumulmultiple.

Figure 3: Statistical distributions of subgroups found on D1.

[†]Server names have been anonymized

Table V: Execution time (in seconds) of SD algorithms.

Algo	**Beam-Search** (heuristic)				**Depth-First** (exhaustive)			
# patterns (k)	10		50		10		50	
depth	2	3	2	3	2	3	2	3
D1	20.17	90.39	113.71	274.60	`limit`	`limit`	`limit`	`limit`
D2	5.35	5.40	28.44	45.04	440.01	`limit`	458.2	`limit`
D3	0.75	0.83	3.94	4.27	0.18	0.50	0.53	0.59
D4	10.73	10.98	56.30	62.35	`limit`	`limit`	`limit`	`limit`

B. Quantitative Analysis

We study the time performance of both exhaustive and heuristic SD algorithms on the four datasets. For each case, we set different values for the number of returned subgroups k, and the depth, i.e, the maximum number of selectors per pattern. Results are provided in Table V. The `limit` value refers to an execution time that exceeds $1,000$ seconds. Beam-Search was able to finish in less than 274 seconds for all configurations, while Depth-First exceeded $1,000$ seconds in many cases. Exceptionally in **D3**, Depth-First took less time than Beam-Search. This may be due to the relatively small number of features compared to other datasets. These results also show the impact of parameters, i.e., the number of returned patterns and the depth. In fact, the higher they are, the longer is the execution time. In our qualitative experiments, we use Beam-Search with a beam width of 50 for D1 and D4, while we exploit results from Depth-First on D2 and D3.

C. Interactive Subgroup Discovery

Figure 4 illustrates our interactive visualisation: it can manage different data types, both for input features as well as the target, including nominal and numerical attributes. It also provides a range of interestingness measures and algorithms. Its main window consists of 3 important panels: (1) the dataset properties, (2) the search strategy, and (3) the results.

Dataset panel. It allows the user to select a subset of data of interest thanks to data points selection where queries are plotted, e.g., w.r.t. execution times and row counts, but also filters on query properties. The graph on the right simply rescale the selection made on the left graph.

Search strategy panel. This panel enables the configuration of the mining task. First, the user needs to define the target. There are two possibilities: (1) choose a specific attribute as target, or (2) graphically create a binary target by associating its positive class to the data subset selected in the right graph, and the remaining data of the left graph as negative class. After that, she can specify the interestingness as well as the mining algorithm. Finally, it remains to set the desired number of returned subgroups, and the maximum depth of pattern-rules.

Results. Once the mining task is executed, this panel shows the identified subgroups. For each, it displays the corresponding pattern along with relevant statistics such as the subgroup size, the median, etc.

V. Related Work

For decades, extracting interesting patterns from query workloads has been of great importance in database research. A variety of related methods have been proposed to perform specific tasks on workloads. The use case most closely related to our solution is performance analysis [2], [11], [26]. However, most of these approaches used clustering-based methods which are not practical to identify subsets of data that specifically discriminate a property of interest. On the other hand, Several major commercial database systems have developed tools to automate this task such as query planner and optimizer; Microsoft SQL Server has included the index selection feature as part of its Tuning Advisor since SQL Server 2000 [39]. Even if these tools are widely used by DBAs, they remain specific and non-generic tools as they are limited to certain features. Indeed, using query optimizer for example, requires digging into individual cases to figure out the issue in each query separately, while a Subgroup Discovery approach aims to identify issues for a *subset* of queries sharing some specific properties w.r.t any user-defined target. Moreover, it can be argued that Subgroup Discovery may be used to assist query planner with very specific cases i.e., providing interesting cases to be investigated with query optimizer. We are the first to address the challenging task of adapting Subgroup Discovery for a complex and generic Workload Analysis problem. In what follows, we describe in more details the different Workload analysis tasks that have been studied in the literature.

Performance Optimization. Database system performance can be tuned by recommending the appropriate set of indexes to speed up query processing. However the complexity of index selection grows quadratically with the workload size [26]. Therefore, several approaches [1], [2], [11], [26] tackle this challenge by finding a compressed workload that is highly representative i.e., a smaller substitute workload that has similar performance characteristics as the original workload. This compression problem is NP-Hard [2]. Thus, existing approaches have used a variety of heuristic techniques ranging from random sampling [2] and clustering [1] to the use of sophisticated Machine Learning models [11]. For instance, [1], [2] use a distance function that measures the difference between pairs of SQL statements, with respect to the workload-driven task. Then, they propose multiple summarization techniques including K-Medoids, random sampling and all pairs greedy algorithm. More recently, [40] propose query structure based clustering algorithms that rely only on the syntactic information of the query. Query2Vec [11] cluster queries based on representations computed using several NLP approaches.

Insider threats identification. [14] propose a semi-supervised approach to analyse database access patterns. It starts by clustering SQL queries using a similarity function that is defined over query structures. Some of these clusters are labeled by experts as potential security threats. Then, these labels are used to generate patterns that enable the automatic classification of remaining clusters.

Query Recommendation. This task aims to assist non-expert users by providing them with personalized SQL query recommendations that correspond to their information needs. [3]

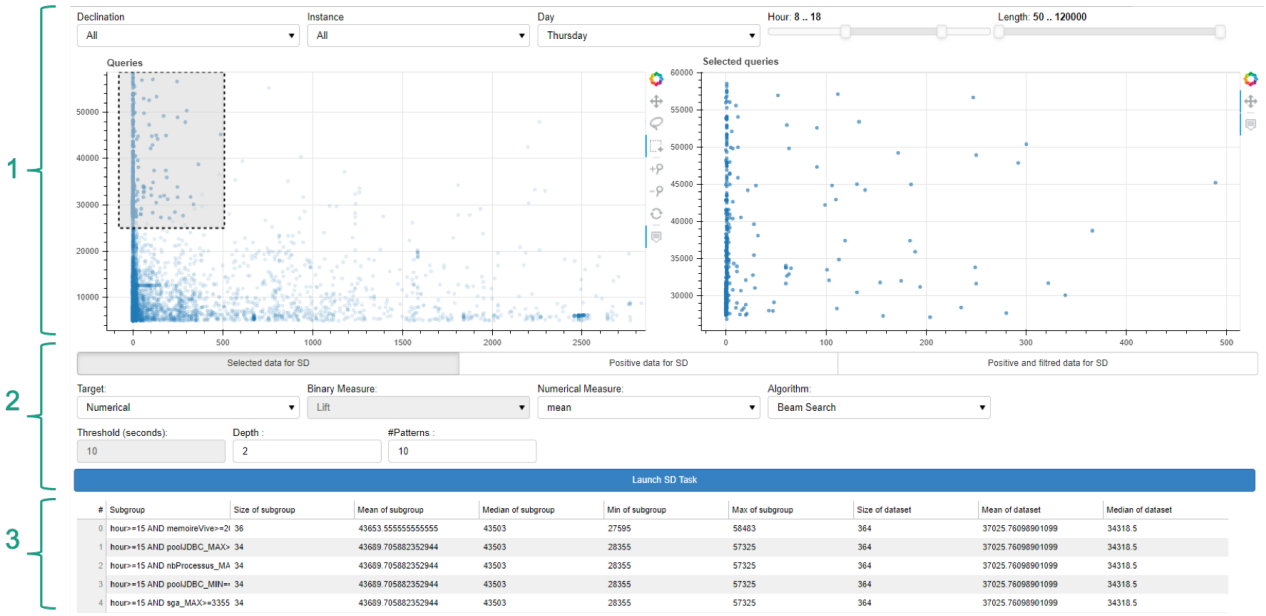


Figure 4: Main sections of the interactive SD tool: (1) dataset properties, (2) search strategy, and (3) results.

exploit a collaborative filtering paradigm where users with similar querying behavior are assumed to be interested in retrieving similar data. [4] introduce an order-sensitive model to compare OLAP user sessions where the order of queries within a session influences the similarity of sessions. ExplIQuE [41] uses clustering and decision trees to extend a given query, by suggesting a set of possible selection predicates to add to the query, that aim at dividing the initial answer set to identify interesting exploration zones.

Finding user interest. This problem aims to track the user's historical querying behavior to seek for her interest. In [12], the authors propose a query similarity metric based on the notion of the so-called access area. This area captures the part of the data space that the user is mostly interested in. Another related work [42] compares queries based on returned results, then cluster the data to help users locate interesting results.

Antipatterns Detection. This problem consists in extracting patterns that generally lead to unnecessary SQL statements which may have a negative effect on performance, or introduce bias on any subsequent workload analysis [5]. In [6], the proposed method analyzes metadata tables to detect *design* antipatterns that reflect errors in the database schema. Whereas [5], [7] are interested in antipatterns related to performance degradation. [7] use static code analysis and rule-based approach, while in [5], the final goal behind cleaning query logs is to simplify the identification of interests of database users.

Visual Analysis. Makiyama et al. [25] use SOM (Self Organizing Map) as a visualization tool due to its quantization and projection properties. The provided visualization gives an idea of the overall shape of the data and helps to detect possible cluster structures in the SQL workload. QueryScope [43] aims to find better tuning opportunities by helping users identify shared patterns between queries while providing a variety of viewing options so that a user can focus on query relevant aspects.

VI. Conclusion and Perspectives

Mining patterns in SQL workloads helps DBAs discovering subgroups of queries sharing some properties and strongly discriminating either a nominal target or a metrics property. We developed a methodology based on Subgroup Discovery which can be tuned in terms of descriptive and target attributes, mining algorithms and pattern quality measures. We proposed a visualisation tool for helping practitioners to make subgroup discovery possible with an interactive platform. We empirically showed how it can elicit hypotheses of interest from queries run on hundreds of databases. We are currently working on integrating our approach in a large scale supervision framework for daily preventive maintenance for our DBA team. Subgroup discovery can be extended in many ways to provide better results. First, we realized through experiments that we often need to consider multiple targets, for example to identify patterns of queries which return few rows while having high running times. Second, although we consider a rich pattern language in comparison to other approaches, we can exploit the syntactic tree structure of the queries, to mine tree patterns, more expressive than conjunctions of SQL clauses. Third, an effort is needed to produce more qualitative subgroup sets with more diversity (data cover) and less redundancy [44], and directly considering a quality measure on the subgroup set, turning the *top-k mining problem* into *subgroup set mining* [45]. Finally, subjectivness and practitioner preferences should be considered by the mining algorithms through an interactive discovery process. These are actually current challenges in the field of subgroup discovery.

REFERENCES

[1] S. Chaudhuri, V. Narasayya, and P. Ganesan, "Primitives for workload summarization and implications for SQL," in *Proceedings 2003 VLDB Conference*. Elsevier, 2003, pp. 730–741.

[2] S. Chaudhuri, A. K. Gupta, and V. Narasayya, "Compressing SQL workloads," in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, 2002, pp. 488–499.

[3] J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. S. V. Varman, "SQL QueRIE recommendations," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1597–1600, 2010.

[4] J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, and E. Turricchia, "Similarity measures for OLAP sessions," *Knowledge and information systems*, vol. 39, no. 2, pp. 463–489, 2014.

[5] N. Arzamasova, M. Schäler, and K. Böhm, "Cleaning antipatterns in an SQL query log," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 3, pp. 421–434, 2017.

[6] E. Eessaar, "On query-based search of possible design flaws of SQL databases," in *Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering*. Springer, 2015, pp. 53–60.

[7] T.-H. Chen, W. Shang, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora, "Detecting performance anti-patterns for applications developed using object-relational mapping," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 1001–1012.

[8] Q. T. Tran, K. Morfonios, and N. Polyzotis, "Oracle workload intelligence," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1669–1681.

[9] P. S. Yu, M. Chen, H. Heiss, and S. Lee, "On workload characterization of relational database environments," *IEEE Trans. Software Eng.*, vol. 18, no. 4, pp. 347–355, 1992.

[10] G. Kul, D. T. A. Luong, T. Xie, V. Chandola, O. Kennedy, and S. Upadhyaya, "Similarity metrics for SQL query clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 12, pp. 2408–2420, 2018.

[11] S. Jain, B. Howe, J. Yan, and T. Cruanes, "Query2vec: An evaluation of NLP techniques for generalized workload analytics," *arXiv preprint arXiv:1801.05613*, 2018.

[12] N. Arzamasova, K. Böhm, B. Goldman, C. Saaler, and M. Schäler, "On the usefulness of SQL-query-similarity measures to find user interests," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 10, pp. 1982–1999, 2019.

[13] X. Yang, C. M. Procopiuc, and D. Srivastava, "Summarizing relational databases," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 634–645, 2009.

[14] G. Kul, D. Luong, T. Xie, P. Coonan, V. Chandola, O. Kennedy, and S. Upadhyaya, "Ettu: Analyzing query intents in corporate databases," in *Proceedings of the 25th international conference companion on world wide web*, 2016, pp. 463–466.

[15] Z. Zolaktaf, M. Milani, and R. Pottinger, "Facilitating SQL query composition and analysis," in *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, 2020, pp. 209–224.

[16] W. Klösgen, "Explora: A multipattern and multistrategy discovery assistant," in *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996, pp. 249–271.

[17] M. Atzmueller, "Subgroup discovery," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 5, no. 1, pp. 35–49, 2015.

[18] B. R. Goldsmith, M. Boley, J. Vreeken, M. Scheffler, and L. M. Ghiringhelli, "Uncovering structure-property relationships of materials by subgroup discovery," *New Journal of Physics*, vol. 19, no. 1, p. 013031, 2017.

[19] S. Helal, J. Li, L. Liu, E. Ebrahimie, S. Dawson, and D. J. Murray, "Identifying key factors of student academic performance by subgroup discovery," *Int. J. Data Sci. Anal.*, vol. 7, no. 3, pp. 227–245, 2019.

[20] C. C. Licon, G. Bosc, M. Sabri, M. Mantel, A. P. Fournel, C. Bushdid, J. Golebiowski, C. Robardet, M. Plantevit, M. Kaytoue, and M. Bensafi, "Chemical features mining provides new descriptive structure-odor relationships," *PLoS Comput. Biol.*, vol. 15, no. 4, 2019.

[21] Mozilla, "Moz SQL parser." [Online]. Available: https://github.com/mozilla/moz-sql-parser

[22] Oracle help center, "Active session history." [Online]. Available: https://docs.oracle.com/database/121/REFRN/GUID-69CEA3A1-6C5E-43D6-982C-F353CD4B984C.htm

[23] pgsentinel, "pgsentinel – sampling active session history." [Online]. Available: https://github.com/pgsentinel/pgsentinel

[24] K. Aouiche, P.-E. Jouve, and J. Darmont, "Clustering-based materialized view selection in data warehouses," in *East European conference on advances in databases and information systems*. Springer, 2006, pp. 81–95.

[25] V. H. Makiyama, J. Raddick, and R. D. Santos, "Text mining applied to SQL queries: A case study for the sdss skyserver." in *SIMBig*, 2015, pp. 66–72.

[26] S. Deep, A. Gruenheid, P. Koutris, J. F. Naughton, and S. Viglas, "Comprehensive and efficient workload compression," *Proc. VLDB Endow.*, vol. 14, no. 3, pp. 418–430, 2020.

[27] T. Scheffer and S. Wrobel, "Finding the most interesting patterns in a database quickly by using sequential sampling," *J. Mach. Learn. Res.*, vol. 3, pp. 833–862, 2002.

[28] N. Lavrac, B. Kavsek, P. A. Flach, and L. Todorovski, "Subgroup discovery with CN2-SD," *J. Mach. Learn. Res.*, vol. 5, pp. 153–188, 2004.

[29] F. Lemmerich, "Novel techniques for efficient and effective subgroup discovery," Ph.D. dissertation, Julius Maximilians University Würzburg, Germany, 2014.

[30] B. F. Pieters, A. Knobbe, and S. Dzeroski, "Subgroup discovery in ranked data, with an application to gene set enrichment," in *Proceedings preference learning workshop (PL 2010) at ECML PKDD*, vol. 10, 2010, pp. 1–18.

[31] F. Lemmerich, M. Atzmueller, and F. Puppe, "Fast exhaustive subgroup discovery with numerical target concepts," *Data Min. Knowl. Discov.*, vol. 30, no. 3, pp. 711–762, 2016.

[32] F. Lemmerich and M. Becker, "pysubgroup: Easy-to-use subgroup discovery in python," in *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, vol. 11053. Springer, 2018, pp. 658–662.

[33] P. Clark and T. Niblett, "The CN2 induction algorithm," *Mach. Learn.*, vol. 3, pp. 261–283, 1989.

[34] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, "Using of jaccard coefficient for keywords similarity," in *Proceedings of the international multiconference of engineers and computer scientists*, vol. 1, no. 6, 2013, pp. 380–384.

[35] M. Atzmüller and F. Puppe, "Semi-automatic refinement and assessment of subgroup patterns," in *Proceedings of the 21st International Florida Artificial Intelligence Research Society Conference*.

[36] S. Wrobel, D. Wettschereck, E. Sommer, and W. Emde, "Extensibility in data mining systems," in *KDD*. AAAI Press, 1996, pp. 214–219.

[37] W. Klösgen and M. May, "Spatial subgroup mining integrated in an object-relational spatial database," in *PKDD*, ser. LNCS, vol. 2431. Springer, 2002, pp. 275–286.

[38] M. Atzmueller and F. Lemmerich, "VIKAMINE - open-source subgroup discovery, pattern mining, and analytics," in *ECML/PKDD (2)*, ser. LNCS, vol. 7524. Springer, 2012, pp. 842–845.

[39] S. Agrawal, S. Chaudhuri, L. Kollár, A. P. Marathe, V. R. Narasayya, and M. Syamala, "Database tuning advisor for microsoft SQL server 2005: demo," in *SIGMOD Conference*. ACM, 2005, pp. 930–932.

[40] T. Xie, O. Kennedy, and V. Chandola, "Query log compression for workload analytics," *arXiv preprint arXiv:1809.00405*, 2018.

[41] M. L. Guilly, J. Petit, V. Scuturici, and I. F. Ilyas, "Explique: Interactive databases exploration with SQL," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*. ACM, 2019, pp. 2877–2880.

[42] Z. Chen and T. Li, "Addressing diverse user preferences in SQL-query-result navigation," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 2007, pp. 641–652.

[43] L. Hu, K. A. Ross, Y.-C. Chang, C. A. Lang, and D. Zhang, "Queryscope: visualizing queries for repeatable database tuning," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1488–1491, 2008.

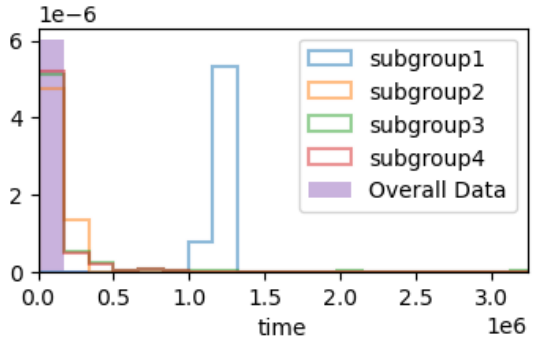
[44] G. Bosc, J. Boulicaut, C. Raïssi, and M. Kaytoue, "Anytime discovery of a diverse set of patterns with monte carlo tree search," *Data Min. Knowl. Discov.*, vol. 32, no. 3, pp. 604–650, 2018.

[45] A. Belfodil, A. Belfodil, A. Bendimerad, P. Lamarre, C. Robardet, M. Kaytoue, and M. Plantevit, "FSSD - A fast and efficient algorithm for subgroup set discovery," in *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2019, pp. 91–99.

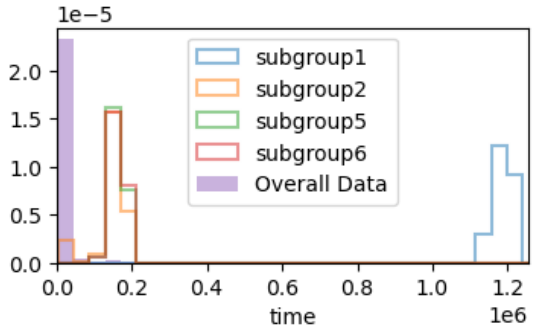
APPENDIX

A. Comparison between interestingness measures

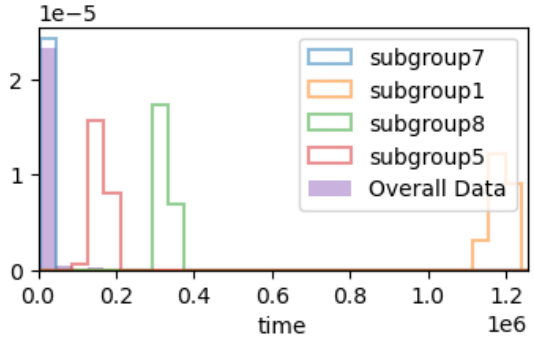
In this section, we aim at comparing the resulting subgroup patterns when using different measures of interest. In particular, our study is conducted on the two examples D1 and D2 defined previously in the Section IV. In the first example D1, we chose the `median` instead of the `mean` because the `mean` is very sensitive to outliers. The results found for each measure are presented in Table VI. Note that we can obtain similar subgroup patterns for different interestingness measures as shown in Figure 5. For instance, the subgroup (s_1) is always identified, regardless of the measure of interest used for the subgroup discovery approach. Different from the `median`-based approach, the two subgroups (s_3) and (s_4) are identified as interesting subgroups w.r.t. to the `mean` measure. However, by analysing their distributions, we do not observe a significant divergence from the original distribution compared to (s_5) and (s_6). Indeed, the subgroups (s_3) and (s_4) contain some slow queries but still also contain many queries that execute quickly. On the other hand, the `T-score` based approach incorporates the mean and the standard deviation of the target value to reflect the cohesion of the subgroup. Except the subgroup (s_3) that has a `support` of 45, identified subgroups are small in size. This does not match the assumption on this measure which requires the subgroup to contain at least 30 objects. In the second example, we show that frequent patterns that rely only on the `support` measure, are not always interesting. For example, the subgroup (s_{10}) contains slow queries that takes more than 10 seconds to execute, but its precision is poor compared to `Lift` and `WRAcc` measures. The `Lift` measure evaluates the subgroups based only on the precision. Usually with lift, we end up with small subgroups but with high precision. In contrast, `WRAcc` and `Binomial` measures depend on the subgroup size. This means that one subgroup can be prioritized over another subgroup that has more precision but contains far fewer objects.



(a) `mean`.



(b) `median`.



(c) T-score.

Figure 5: Statistical distributions of subgroups found on D1 for different measures.

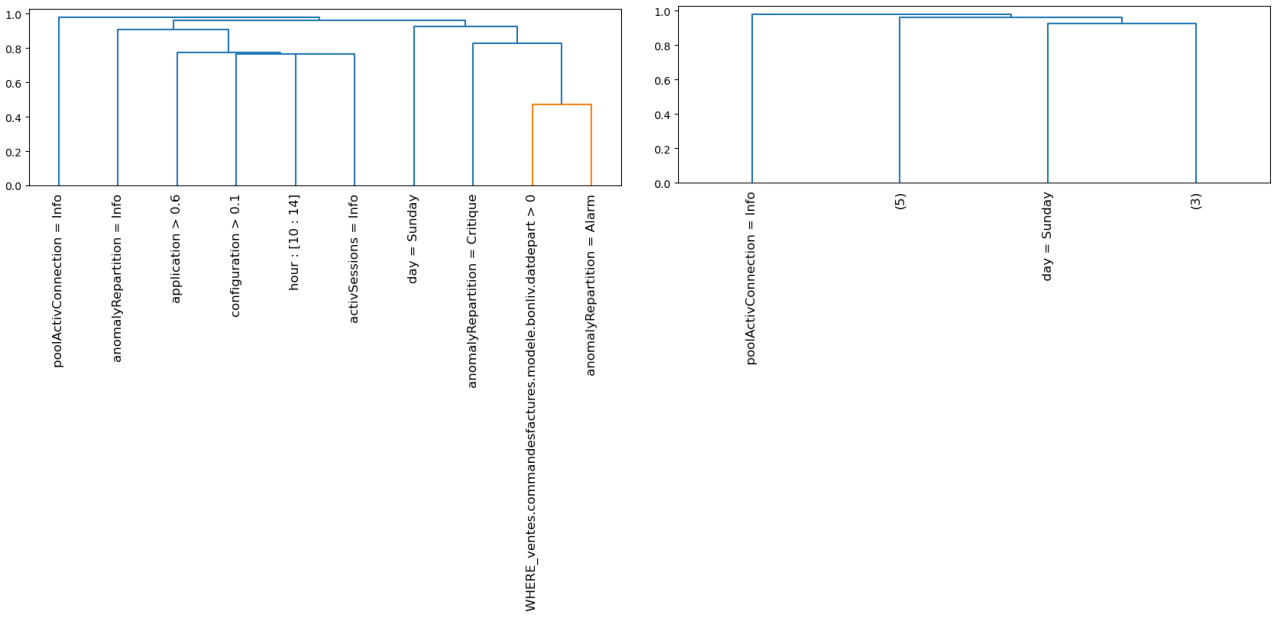
B. Effectiveness of the postprocessing phase

When selecting interesting subgroups based only on the measure of interest, we may end up with redundant subgroups. In this section, we show how the postprocessing phase is useful in providing the user with interesting but diverse patterns. We use the Agglomerative Hierarchical clustering on example D4 to extract the most representative subgroups that correlates with the alert *blockedSessions*. In the example shown in Figure 6, we initially extract the top-10 subgroups using the `Binomial` Measure. We then perform the clustering on the 10 found subgroups based on the `Jaccard distance`. Afterwards, we chose a partition of (4) different subgroups by truncating the tree at the distance (0.91). This means that we allow only subgroups having at most 0.09 of similarity between them. We end up with 2 subgroups patterns and two clusters that contains 5 and 3 subgroups respectively. For each cluster we choose the best subgroup w.r.t to its measure of interest, thus we end up with the two patterns that we displayed in Section IV.

Table VI: Subgroup Discovery Results with different measures

ID	Target	Measure	Subgroup patterns	Size	Quality
D1	**time** (Numerical)	mean	(s_1) : WHERE_stocks.gestion.modele.lot.prod.ref.auditinfo.etat > 0	8	$78 \times$ mean_*dataset*
			(s_2) : FROM_ventes.cumuls.modele.cumulmultiple > 0	451	$9 \times$ mean_*dataset*
			(s_3) : WHERE_stocks.gestion.modele.mvtrealise.flagaepurer > 0	602	$6 \times$ mean_*dataset*
			(s_4) : SELECT_stocks.gestion.modele.mvtrealise.ik > 0	719	$6 \times$ mean_*dataset*
		median	(s_5) : WHERE_ventes.cumuls.modele.cumulmultiple.valzvcli**x** > 0	45	$21 \times$ *med_dataset*
			(s_6) : WHERE_.ventes.cumuls.modele.cumulmultiple.valzvart**x** > 0	45	$21 \times$ *med_dataset*
		T-score	(s_7) : WHERE_stocks.achats.cadencier_fournisseur.modele.cadencier.mat.art.ik > 0	8	-
			(s_8) : WHERE_achats.fournisseurs.modele.fourlivperiodereg.datfin > 0	2	-
D2	**slow** (Binary) $prec \simeq 60\%$	Support	(s_9) : WHERE_stocks.gestion.modele.mvtrealise.lot.ik > 0	29827	$prec \simeq 65\%$
			(s_{10}) : dbCursorsMax < 2000	30548	$prec \simeq 65\%$
		Lift	(s_{11}) : GROUPBY_stocks.gestion.modele.mvtrealise.refexterne > 0	131	$prec = 100\%$
			(s_{12}) : serverName = serverX $\wedge$ systemI/O > 50	38	$prec = 100\%$
		WRAcc / Binomial	(s_{13}) : WHERE_stocks.gestion.modele.mvtrealise.etatsynchro $> 0 \wedge$ jdbcMax < 200	20668	$prec \simeq 99\%$
			(s_{14}) : WHERE_stocks.gestion.modele.mvtrealise.auditinfo.datcre $> 0 \wedge$ dbVersion $= 2.3$	20675	$prec \simeq 99\%$

[h]



(a) Hierarchical Clustering of all patterns.

(b) Truncated Hierarchical clustering.

Figure 6: **Post processing.** Hierarchical clustering based on Jaccard distance