User-Guided Discovery of Patterns in Sequential Data

Hannes Ueck
Supervised by Matthias Weidlich
Hasso-Plattner-Institut
Potsdam, Germany
hannes.ueck@hpi.de

ABSTRACT

Identifying patterns that characterize situations of interest in event data is a recurring challenge in domains such as system monitoring or anomaly detection. While domain experts can often recognize such situations, they rarely know the entire underlying pattern that describes them. We study the problem of discovering such patterns as queries in the `MATCH_RECOGNIZE` query model, which requires discovering event definitions and their temporal structure jointly. This distinguishes our setting from classical frequent pattern mining, where event types are assumed to be predefined. We show that discovery for `MATCH_RECOGNIZE` is challenging because event definitions and temporal patterns are interdependent and cannot be discovered in isolation. This interdependence enlarges the search space. Moreover, refining a candidate query may increase or decrease the number of matches, which limits monotonicity-based pruning. Based on our analysis, we outline the following research directions: iterative pipeline-based query discovery, heuristic search over the full query space guided by upfront user input, and adaptive search that incorporates iterative user feedback.

VLDB Workshop Reference Format:
Hannes Ueck. User-Guided Discovery of Patterns in Sequential Data. VLDB 2026 Workshop: VLDB Ph.D. Workshop.

1 INTRODUCTION

In many settings, systems continuously log event data that can be used to gain insight into processes, machines, users, or software systems [16]. Often, domain experts can identify situations of interest, such as a machine failure in a factory, a straggling data analysis pipeline in a compute cluster, or a user with unusual behavior in an online system. Yet, the knowledge about the exact pattern in the event data that characterizes these situations is often only partial, or missing entirely. Investigating such situations is then a manual and iterative process, in which domain experts inspect large event logs and test patterns against the data.

We want to find the commonalities of the situations marked by the domain expert. The discovered pattern can then support root-cause analysis and reveal which conditions precede a situation of interest. We use the SQL operator `MATCH_RECOGNIZE` as the query model [1]. It defines variables that bind rows based on predicates, along with a pattern over these variables. The query

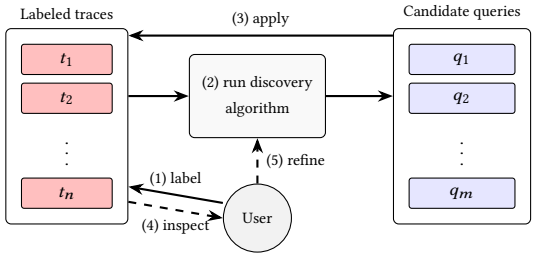

Figure 1: Overview of user-guided query discovery. (1) The user labels traces $t_1, \ldots, t_n$ as situations of interest or background. (2) The discovery procedure produces a set of candidate `MATCH_RECOGNIZE` queries $q_1, \ldots, q_m$. (3) The candidates are applied to the labeled traces. (4) The user inspects the resulting matches and (5) refines the result through feedback.

therefore describes both which rows play the role of pattern events and how these events are ordered. Adopting a standard SQL operator, our discovered queries can be directly deployed in existing database systems without additional tooling. Evaluating the queries over continuously updated event data, they may also detect future occurrences of the situation of interest.

Example 1.1. Consider a compute cluster that logs job scheduling events with attributes `job`, `status` ∈ {`S`, `E`, `F`}, and `memory`. A user notices that some jobs fail and suspects that high memory usage prior to eviction is a common precursor. The following `MATCH_RECOGNIZE` query captures this pattern:

```
SELECT * FROM log
MATCH_RECOGNIZE (
  PARTITION BY job ORDER BY time
  PATTERN (A+ B C)
  DEFINE
    A AS A.status = 'S' AND A.memory > 80,
    B AS B.status = 'E' AND B.memory > A.memory,
    C AS C.status = 'F'
)
```

It matches job traces where one or more scheduling events with high memory usage are followed by an eviction with even higher memory, and then a failure.

While automated discovery of such queries may help a user understand what contributes to a situation of interest, it is inherently difficult. The search space is large and related, less expressive event-query discovery problems are computationally intractable [7]. Specifically, the search space grows with the number of possible patterns, the number of attributes, and the size of the attribute domains. Existing methods for query discovery either assume predefined event types, use less expressive query languages, or search exhaustively, which becomes infeasible even for restricted variants of the problem [14].

Proceedings of the VLDB Endowment. ISSN 2150-8097.

In this work, we formulate the discovery of MATCH_RECOGNIZE queries from labeled situations of interest as a user-guided search problem. Figure 1 gives an overview of the approach, where discovery turns labeled traces into candidate queries, which the user refines through feedback. What makes this problem difficult is that the query defines event definitions and a temporal pattern at the same time, while both influence each other. This coupling, together with the size of the search space, makes enumeration infeasible and limits pruning techniques used in classical sequence mining. In the PhD project, we plan to begin by decomposing discovery into an iterative mining pipeline that selects predicates, forms pattern variables, and mines temporal patterns over them. Later we plan to work on heuristic and interactive search over the full search space.

2 RELATED WORK

Below, we summarize research fields that are related to our work.

Sequential and temporal pattern mining. Sequential pattern mining algorithms discover frequently occurring subsequences in sequence databases [4]. Classical approaches such as PrefixSpan, SPADE, and Apriori-based methods operate on sequences of discrete event types and prune the search space using frequency-based criteria. A closely related framework is episode mining, which discovers frequently occurring ordered or partially ordered collections of events with a sliding time window over a single long sequence [10]. However, all these methods assume that event types are given in advance.

Automated CEP rule discovery. The MATCH_RECOGNIZE query model is similar to pattern detection rules that are evaluated in complex event processing (CEP) over streams of events. In this field, similar discovery problems have been studied. Early approaches for CEP rule discovery solve separate subproblems for event type selection, time window learning, and predicate inference, but assume that event types are predefined [11]. The IL-Miner addresses this limitation by identifying event types using frequent sequence mining before constructing pattern candidates [5]. DISCES proposes algorithms for an exhaustive search of the query candidate space, which is limited to queries with constants and equality constraints [14]. Other approaches use evolutionary algorithms to find CEP rules but assume fixed event types [3]. Approaches that avoid predefined event types include methods that first cluster or label raw data and then mine rules using machine learning methods, such as SVMs or LSTMs [9, 15]. However, our ideas differ as we aim at supporting a richer query model, while also incorporating user input into the search.

Relational constraint discovery. Another line of research discovers constraints from relational data, including functional dependencies, denial constraints, order dependencies, and inclusion dependencies [2, 13]. These methods discover constraints that hold globally over static, unordered relations.

Query synthesis from examples. The related problem of inferring SQL select-project-join (SPJ) queries from examples has been studied under the names of query reverse engineering and query-by-example [12]. These approaches take positive and negative example tuples, or input-output table pairs, and synthesize SPJ queries that reproduce the given examples.

Interactive pattern mining. Interactive approaches to pattern mining involve the user in the exploration of the discovery results. A prominent example for these techniques is MIME, which supports interactive exploration of pattern mining results, including frequent itemsets, association rules, and sequential patterns [6]. Yet, it does not incorporate user feedback in the actual discovery process. Query-from-example systems, in turn, study iterative construction of relational SQL queries from user-provided examples [8]. Existing approaches focus on traditional query models that do not incorporate patterns over row variables, as found in MATCH_RECOGNIZE.

3 PROBLEM DESCRIPTION

Data model. We consider event data stored in a relation with schema $R(A_1, \ldots, A_n)$, where each attribute A_i has a finite domain $\mathtt{dom}(A_i)$. The data is partitioned and ordered by designated attributes — for example, by job identifier and timestamp as in Example 1.1 — yielding a set of ordered sequences of tuples called *traces*. We assume the partitioning and ordering to be given and denote the set of all traces by Σ.

Query model. MATCH_RECOGNIZE [1] is a standard SQL operator that searches each trace for contiguous subsequences matching a row pattern. The operator is structured around three key clauses. The PATTERN clause specifies the structure of the match as a regular expression over named variables, supporting concatenation ($A\,B$), alternation ($A \mid B$), and repetition (A^*, A^+, $A\{m, n\}$). The DEFINE clause assigns a predicate to each variable, which determines which rows may be matched to it. The MEASURES clause specifies which attributes of the match are returned as output. In its full form, predicates may additionally reference aggregate functions over previously matched rows, and the operator supports options for controlling how matches overlap and where the evaluation continues once a match has been found in a trace.

In this work, we consider a fragment of MATCH_RECOGNIZE that retains the core functionality while allowing for automated discovery. We support concatenation, alternation, and repetition of individual variables, but exclude nested compound pattern expressions. For the DEFINE clause, we restrict predicates to conjunctions of atomic comparisons: either a comparison between a variable attribute and a constant ($X.A_i\,\theta\,c$), or a comparison between attributes of two variables ($X.A_i\,\theta\,Y.A_j$), where $\theta \in \{=, <, >\}$. Predicates with disjunction, negation, aggregates, and user-defined functions are not part of this fragment. We refer to the first form as **unary predicates** and to the second as **binary predicates**, since they constrain one and two variables respectively. In Example 1.1, $A.\mathtt{memory} > 80$ is a unary predicate and $B.\mathtt{memory} > A.\mathtt{memory}$ is a binary predicate. We adopt contiguous match semantics, where consecutive pattern variables bind to consecutive rows of the traces.

We model a **query** in this fragment as a tuple $q = (V, P, \Phi)$, where $V = \{X_1, \ldots, X_r\}$ is a finite set of **pattern variables**, P is a pattern over V, and $\Phi = \{\phi_X \mid X \in V\}$ assigns a predicate to each variable. We denote the universe of all such **queries** by $\mathcal{Q}$.

User input. The user labels a subset of traces as **situations of interest** $\mathcal{I} \subseteq \Sigma$. The remaining traces form the **background** $\mathcal{B} = \Sigma \setminus \mathcal{I}$ by default, assuming unlabeled traces are not of interest. The user may instead provide $\mathcal{B}$ explicitly, e.g., when only part of Σ has been labeled. We say a query q matches a trace s if s

contains a contiguous subsequence matched by q. A useful query should match the situations in $\mathcal{I}$ while matching as little of $\mathcal{B}$ as possible. The user may additionally provide **guidance** G, either upfront or between discovery rounds. Hard constraints, such as allowed attributes or pattern structures, restrict the candidate space, whereas soft preferences, such as preferred attributes or query simplicity, affect candidate ranking. Feedback on proposed queries or their matches may update this guidance in subsequent rounds.

Problem 1 (User-Guided Query Discovery)**.** Given a set of traces Σ, situations of interest $\mathcal{I} \subseteq \Sigma$, background $\mathcal{B} \subseteq \Sigma$ with $\mathcal{I} \cap \mathcal{B} = \emptyset$, and user guidance G, find a candidate set $\mathcal{Q}'_G \subseteq \mathcal{Q}$ of executable queries such that:

- each $q \in \mathcal{Q}'_G$ satisfies $\text{supp}(q, \mathcal{I}) \geq \tau$ and $\text{supp}(q, \mathcal{B}) \leq \tau'$ for support thresholds τ, τ', where
$$\text{supp}(q, S) = \frac{|\{s \in S \mid q \text{ matches } s\}|}{|S|},$$
- each $q \in \mathcal{Q}'_G$ satisfies the hard constraints in G, and
- $|\mathcal{Q}'_G| \leq k$ for a budget k, i.e., the result is sufficiently small to be inspected and refined by the user.

4 CHALLENGES

To understand why query discovery for `MATCH_RECOGNIZE` is hard, consider a naive approach that enumerates all candidate queries and tests them against the data. Such an algorithm would generate possible row patterns, assign predicates to the variables in these patterns, and then keep the queries that match the situations of interest while matching little of the background traces. Algorithm 1 sketches this idea. The algorithm illustrates several challenges that we explain in the following.

We bound the enumeration by considering only flat patterns with at most L variable occurrences and repetition bounds at most R. Moreover, the unary and binary predicate-assignment loops jointly assign at most C atomic predicates to each variable, using constants from a finite candidate set.

Algorithm 1: Exhaustive query discovery

Input: Traces Σ, situations of interest $\mathcal{I}$, background $\mathcal{B}$, support thresholds τ, τ'
Output: Set of candidate queries $\mathcal{Q}'$
$\mathcal{Q}' \leftarrow \emptyset$;
forall *patterns P over variables $V = \{X_1, \ldots, X_r\}$* **do**
  forall *unary predicate assignments Φ_u over V* **do**
    forall *binary predicate assignments Φ_b over pairs of variables in P* **do**
      $q \leftarrow (V,\ P,\ \Phi_u \cup \Phi_b)$;
      if $\text{supp}(q, \mathcal{I}) \geq \tau$ ***and*** $\text{supp}(q, \mathcal{B}) \leq \tau'$ **then**
        $\mathcal{Q}' \leftarrow \mathcal{Q}' \cup \{q\}$;
return $\mathcal{Q}'$;

Challenge 1: Search space size. A candidate query combines a row pattern with predicates for each variable in the pattern. The number of row patterns grows quickly with the number of variables due to concatenation, repetition, and alternation. For each pattern, the predicate space is exponential in the number of attributes and domain size, since each variable can be assigned any conjunction of atomic predicates over all attributes and constants. Even fixing the pattern to $A^+\,B\,C$ as in Example 1.1, the threshold in $A.\texttt{memory} > 80$ could be any value in $\texttt{dom}(\texttt{memory})$, and there are many other possible attribute conditions and inter-variable comparisons to consider. Prior work has shown that related, less expressive event-query discovery problems are already computationally hard [7].

Challenge 2: Non-monotone pattern changes. In classical sequence mining, refinement is monotone. That means that adding an event type to a sequence can only reduce the set of matching sequences, enabling pruning of the search space. In `MATCH_RECOGNIZE`, this holds only for predicate-level modifications. Adding a predicate to a variable constrains which rows it can match and can only reduce support. Adding a variable to a pattern, however, can increase or decrease the support. For instance, inserting an unconstrained variable Z between B and C in the pattern $A^+\,B\,C$ of Example 1.1 requires an additional matched event, but also allows the predicates on B and C to be satisfied across a wider span of the trace, which can match traces that $A^+\,B\,C$ did not. Pruning is therefore possible for predicate discovery but not for the pattern space.

Challenge 3: Coupling of pattern and predicates. The row pattern and the predicates cannot be treated as independent subproblems. A pattern only has meaning together with the predicates that define its variables, while a predicate may be useful only because its variable occurs in a particular position. In Example 1.1, the predicate $A.\texttt{memory} > 80$ is meaningful because A occurs before the failure event C, while the same predicate on a variable after the failure would not characterize the precursor pattern. Unary predicates, such as $A.\texttt{memory} > 80$ in Example 1.1, constrain one variable in isolation, but binary predicates, such as $B.\texttt{memory} > A.\texttt{memory}$, constrain the relationship between two matched variables and require the pattern to be known before they can be enumerated. A discovery algorithm may still generate predicates, variables, and patterns in separate steps, but their usefulness has to be judged in the context of complete queries.

Changing the order in which patterns, unary predicates, and binary predicates in Algorithm 1 are generated does not remove the dependency between the pattern and the binary predicates. If binary predicates are generated before the pattern is fixed, the algorithm has to keep track of which later patterns contain the variables referenced by these predicates. If the pattern is generated first, binary predicates can be restricted to variable pairs it contains.

Challenge 4: Matching complexity. Even for a single fixed candidate query, checking whether it matches a trace is computationally expensive. In Example 1.1, the repetition A^+ means the number of ways the pattern can match a trace grows with the trace length. Previous work on a less expressive query model already stated that the matching is the main bottleneck in the search [14]. Since a discovery algorithm must evaluate many candidate queries, this cost increases with the size of the search space and makes exact evaluation of all candidate queries impractical.

5 RESEARCH DIRECTIONS

Based on the preceding analysis, we pursue three alternatives to exhaustive enumeration.

Direction 1: Iterative pipeline-based query discovery. As a first research direction, we study how discovery of `MATCH_RECOGNIZE` queries can be decomposed into an iterative mining pipeline. The pipeline alternates between selecting atomic predicates, combining them into pattern variables, mining the temporal patterns over the variables, and using the obtained information from the matches to adapt the predicate selection in the next iteration. We have implemented an initial prototype that realizes one pass through this pipeline. The prototype enumerates equality predicates for categorical attributes and quantile-based thresholds for numerical attributes. It combines frequent predicate sets into candidate pattern variables, expands possible row-to-variable assignments, mines frequent variable sequences over the resulting labeled traces, and searches for binary predicates between adjacent variables. This prototype provides a starting point for integrating and evaluating further ideas, such as deriving initial predicates using the contrast information between the situations of interest and the background. It also shows that the initial set of predicates affects quality and runtime and should therefore be carefully chosen and refined iteratively rather than passed through the pipeline all at once.

Direction 2: Heuristic search over complete queries. The second direction explores how to search the combined space of patterns and predicates without exhaustive enumeration. Since pattern modifications are non-monotone, a scoring function cannot prune branches the way it would in a monotone space and must instead guide which candidates to explore next. We intend to employ beam search, where a fixed set of best-scoring candidate queries is expanded by local modifications each round and the best results are kept for the next. Since exact matching is expensive, we plan to use approximate support estimates rather than exact evaluation at every step. Predicate modifications are monotone, which enables reusing intermediate match results when a beam step only tightens a predicate. We plan to incorporate a diversity criterion into the scoring function so that the output covers diverse parts of the query space. Additionally, we want to explore options for upfront user input such as relevant attributes, predicate hints, or an initial pattern structure to restrict the search space.

Direction 3: Learning-guided search with iterative user feedback. Unlike the previous direction, the scoring function here is not fixed upfront but learned from user feedback over multiple rounds. In each round, the algorithm presents candidate queries and the user provides feedback, either at the query level by ranking or accepting and rejecting candidates, or at the match level by marking specific matches as correct or incorrect. This allows the algorithm to incorporate preferences that are hard to specify in advance. We plan to investigate reinforcement learning, where feedback acts as a reward signal shaping a search policy, and genetic algorithms, where feedback drives mutation and selection of a query population. Both approaches must handle inconsistent feedback, since users can change their assessment between rounds or give contradictory responses.

6 CONCLUSION

We framed the discovery of `MATCH_RECOGNIZE` queries as a user-guided search problem over event definitions and temporal patterns. The difficulty is that both parts of the query have to be discovered jointly, making the search space too large to exhaustively enumerate. We therefore first decompose discovery into an iterative pipeline, before using heuristics and user guidance to make search over complete queries tractable.

ACKNOWLEDGMENTS

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 414984028 – SFB 1404 FONDA.

REFERENCES

[1] ISO/IEC JTC 1/SC 32. 2016. *ISO/IEC TR 19075-5:2016 Information technology — Database languages — SQL Technical Reports Part 5: Row Pattern Recognition in SQL*. https://www.iso.org/standard/65143.html

[2] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *VLDB J.* 24, 4 (2015), 557–581. https://doi.org/10.1007/S00778-015-0389-Y

[3] Giulio Appetito, Eric Medvet, and Vincenzo Gulisano. 2025. Automated Discovery of CEP Applications with Evolutionary Computing. In *Proceedings of the 19th ACM International Conference on Distributed and Event-based Systems, DEBS 2025, Gothenburg, Sweden, June 10-13, 2025*. ACM, 33–38. https://doi.org/10.1145/3701717.3730548

[4] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Rage Uday Kiran, Yun Sing Koh, and Rincy Thomas. 2017. A survey of sequential pattern mining. *Data Science and Pattern Recognition* 1, 1 (2017), 54–77.

[5] Lars George, Bruno Cadonna, and Matthias Weidlich. 2016. IL-Miner: Instance-Level Discovery of Complex Event Patterns. *Proc. VLDB Endow.* 10, 1 (2016), 25–36. https://doi.org/10.14778/3015270.3015273

[6] Bart Goethals, Sandy Moens, and Jilles Vreeken. 2011. MIME: a framework for interactive visual pattern mining. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, Chid Apté, Joydeep Ghosh, and Padhraic Smyth (Eds.). ACM, 757–760. https://doi.org/10.1145/2020408.2020529

[7] Sarah Kleest-Meißner, Rebecca Sattler, Markus L. Schmid, Nicole Schweikardt, and Matthias Weidlich. 2022. Discovering Event Queries from Traces: Laying Foundations for Subsequence-Queries with Wildcards and Gap-Size Constraints. In *25th International Conference on Database Theory, ICDT 2022, Edinburgh, UK (Virtual Conference), March 29 - April 1, 2022 (LIPIcs)*, Dan Olteanu and Nils Vortmeier (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 18:1–18:21. https://doi.org/10.4230/LIPICS.ICDT.2022.18

[8] Hao Li, Chee-Yong Chan, and David Maier. 2015. Query From Examples: An Iterative, Data-Driven Approach to Query Construction. *Proc. VLDB Endow.* 8, 13 (2015), 2158–2169. https://doi.org/10.14778/2831360.2831369

[9] Yuan Liu, Wangyang Yu, Cong Gao, and Minsi Chen. 2022. An Auto-Extraction Framework for CEP Rules Based on the Two-Layer LSTM Attention Mechanism: A Case Study on City Air Pollution Forecasting. *Energies* 15, 16 (2022). https://doi.org/10.3390/en15165892

[10] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. 1997. Discovery of Frequent Episodes in Event Sequences. *Data Min. Knowl. Discov.* 1, 3 (1997), 259–289. https://doi.org/10.1023/A:1009748302351

[11] Alessandro Margara, Gianpaolo Cugola, and Giordano Tamburrelli. 2014. Learning from the past: automated rule generation for complex event processing. In *The 8th ACM International Conference on Distributed Event-Based Systems, DEBS '14, Mumbai, India, May 26-29, 2014*, Umesh Bellur and Ravi Kothari (Eds.). ACM, 47–58. https://doi.org/10.1145/2611286.2611289

[12] Denis Mayr Lima Martins. 2019. Reverse engineering database queries from examples: State-of-the-art, challenges, and research opportunities. *Inf. Syst.* 83 (2019), 89–100. https://doi.org/10.1016/J.IS.2019.03.002

[13] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. 2015. Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. *Proc. VLDB Endow.* 8, 10 (2015), 1082–1093. https://doi.org/10.14778/2794367.2794377

[14] Rebecca Sattler, Sarah Kleest-Meißner, Steven Lange, Markus L. Schmid, Nicole Schweikardt, and Matthias Weidlich. 2025. DISCES: Systematic Discovery of Event Stream Queries. *Proc. ACM Manag. Data* 3, 1 (2025), 32:1–32:26. https://doi.org/10.1145/3709682

[15] Mehmet Ulvi Simsek, Feyza Yildirim Okay, and Suat Ozdemir. 2021. A deep learning-based CEP rule extraction framework for IoT data. *J. Supercomput.* 77, 8 (2021), 8563–8592. https://doi.org/10.1007/S11227-020-03603-5

[16] Tiago Zonta, Cristiano André da Costa, Rodrigo da Rosa Righi, Miromar José de Lima, Eduardo Silveira da Trindade, and Guann-Pyng Li. 2020. Predictive maintenance in the Industry 4.0: A systematic literature review. *Comput. Ind. Eng.* 150 (2020), 106889. https://doi.org/10.1016/J.CIE.2020.106889