PatternInsight: An Online Approach to Complex Pattern Detection over Mobile Data Streams

Xudong Wu, *Member, IEEE*, Yuyang Ren, Zhenhua Li, *Senior Member, IEEE*, Fei Xu, *Member, IEEE*, Yunhao Liu, *Fellow, IEEE*, and Guihai Chen, *Fellow, IEEE*

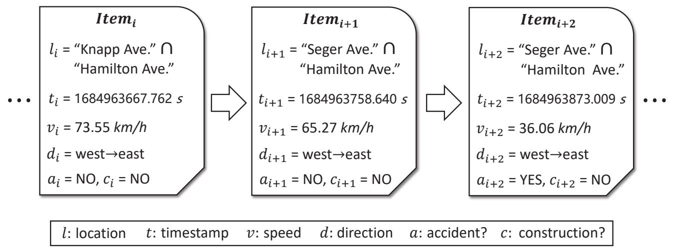
Abstract—Today's mobile applications oftentimes need to detect user-defined complex patterns (*e.g.*, the mysterious "phantom traffic jam") over data streams to support decision making. It is achieved by continuously creating candidate instances that have partially matched a pattern, and meanwhile aggregating common instances (across patterns) for efficiency enhancement. Existing aggregation approaches are taken in a straightforward or intuitive manner, incurring an exponential solution space and thus having to be executed offline. This paper explores how to significantly accelerate aggregation so as to make pattern detection online executable, even suited to the emerging serverless runtime that involves complicated state synchronizations among distributed cloud functions. By comprehensively investigating a wide variety of mobile data streams, we note the existence of a latent hierarchical cluster structure among complex patterns (in terms of their instance similarities), which can be utilized to quickly aggregate common instances without going through the exponential solution space. To extract the latent information, we devise a content-aware structural entropy minimization algorithm to properly determine intra-cluster patterns, together with a lightweight differential compensation mechanism to maintain those inter-cluster "residual" relations among patterns. Evaluations on real-world vehicle and sensor network data streams illustrate that the resulting approach, dubbed PatternInsight, saves the aggregation time by 10× to 50× and reduces the instance size by 40%.

Index Terms—Mobile data stream, Complex pattern detection, Online aggregation, Serverless computing, Structural information theory

1 Introduction

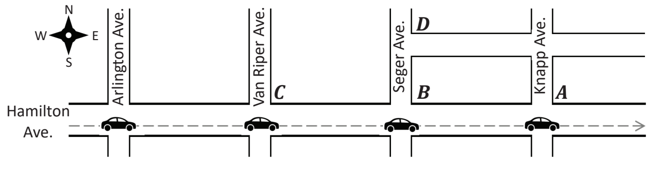
Many mobile applications (*e.g.*, mobile payment and navigation) are featured by generating massive streaming data at high rates, which are required to (or had better) be processed and analyzed in real time [1]–[5]. Traditionally, the data items in streams are handled independently for simple use cases, such as transaction recording and device localization, usually incurring a linear time/space complexity. In recent years, with the development of sophisticated or intelligent mobile applications like IoT-based supply chain, intelligent transport and network management, there emerge intensive requirements for detecting *complex patterns*, which are typically the user-defined combinations of certain data items [6]–[10].

Consider a realistic example of monitoring traffic flows in a city, where a data stream consists of time-ordered vehicle traveling information (as shown in Fig. 1). Here, $\boldsymbol{Item_i}$ records an event that a vehicle passes by the intersection of Knapp Ave. and Hamilton Ave. from west to east, with a speed of 73.55 km/h at 1684963667.762 s (represented by the Linux timestamp plus milliseconds); also, there is no traffic accident or road construction at that time (*i.e.*, a_i= c_i= NO). Note that the monitoring system cannot recognize plate numbers and thus cannot identify vehicles.



$Item_i$
l_i = "Knapp Ave." ∩ "Hamilton Ave."
t_i = 1684963667.762 s
v_i = 73.55 km/h
d_i = west→east
a_i = NO, c_i = NO
$Item_{i+1}$
l_{i+1} = "Seger Ave." ∩ "Hamilton Ave."
t_{i+1} = 1684963758.640 s
v_{i+1} = 65.27 km/h
d_{i+1} = west→east
a_{i+1} = NO, c_{i+1} = NO
$Item_{i+2}$
l_{i+2} = "Seger Ave." ∩ "Hamilton Ave."
t_{i+2} = 1684963873.009 s
v_{i+2} = 36.06 km/h
d_{i+2} = west→east
a_{i+2} = YES, c_{i+2} = NO
l: location t: timestamp v: speed d: direction a: accident? c: construction?

Fig. 1: An example of the data stream in the intelligent transport application.



$$Complex\ pattern\ \mathcal{P}: \{\exists i,j,k\ \ (i \neq j \wedge i \neq k \wedge j \neq k),\ l_i = A, l_j = B, l_k = C,\ t_i < t_j < t_k,\ t_k - t_i \leq 20\ min,\ v_i, v_j, v_k \leq 40\ km/h,\ d_i, d_j, d_k = west \rightarrow east,\ a_i, a_j, a_k = NO,\ c_i, c_j, c_k = NO\}$$

Fig. 2: A real-world example of the complex pattern $\mathcal{P}$ that indicates the probable occurrence of a phantom traffic jam.

With this data stream, traffic jams can usually be automatically and timely detected. Jams caused by accidents or constructions can be simply detected based on the attributes a and c in Fig. 1, but the detection of complex cases like a *phantom traffic jam* (*i.e.*, successive low-speed driving occurs without the occurrence of accidents or constructions) [11] requires combing multiple items. When the roads in Fig. 2 are clear, the speeds of vehicles are ∼70 km/h; if the west-to-east speeds at locations A, B, and C successively drop below 40 km/h within 20 minutes and there is no accident or construction (*i.e.*, the complex pattern $\mathcal{P}$ is detected), a

This work is supported by the National Key R&D Program of China under grant 2022YFB4500703, the National Natural Science Foundation of China under grants 62202266 and 62372184, and the Science and Technology Commission of Shanghai Municipality under grant 22DZ2229004.

- *X. Wu, and F. Xu are with East China Normal University, Shanghai, China. (email: xudongwu1993@gmail.com, fxu@cs.ecnu.edu.cn)*
- *Z. Li, and Y. Liu are with Tsinghua University, Beijing, China. (email: {lizhenhua1983, yunhaoliu}@gmail.com) (Corresponding author: Z. Li)*
- *Y. Ren and G. Chen are with Shanghai Jiao Tong University, Shanghai, China. (email: renyuyang@sjtu.edu.cn, gchen@cs.sjtu.edu.cn)*

phantom traffic jam may have occurred.

In practice, complex pattern detection is crucial to many other mobile applications like supply chain [12], mobile financial services [13], and app operations [14], where the complex patterns can be RFID movement trajectories, abnormal transaction behaviors, and app crash distributions.

1.1 Motivation

Complex pattern detection is typically performed by a complex event processing (CEP) system [13], [15], [16] deployed at back-end servers/ edge devices, or executed by serverless functions. Monitoring an input data stream, a CEP system continuously creates various combinations of incoming data items as *candidate instances* that have partially matched one or more complex patterns (*e.g.,* creating the combinations of data items generated at locations A, B, and C as the candidate instances of $\mathcal{P}$). Meanwhile, it examines whether these instances satisfy other requirements of complex patterns, such as item sequences, time intervals, and attribute constraints [17]–[19]; the eligible instances are then stored in the memory.

In early-stage CEP systems, the instances of each complex pattern are independently created and stored [17], [18], [20]. Consider an example of parallelly detecting two complex patterns. One is the complex pattern $\mathcal{P}$ in Fig. 2; the other, denoted as $\mathcal{P}'$, indicates the probable occurrence of a phantom jam spreading from Hamilton Ave. to Seger Ave., as shown below:

$$\begin{gathered}\textit{Complex pattern } \mathcal{P}'\!:\{\,\exists i,j,k\ \ (i \neq j \land i \neq k \land j \neq k),\ \ l_i = A, l_j = B, l_k = D,\\ t_i < t_j < t_k,\ \ t_k - t_i \leq 20\ min,\ \ v_i, v_j, v_k \leq 40\ km/h,\\ d_i, d_j = west \rightarrow east, d_k = north \rightarrow south,\quad a_i, a_j, a_k = NO,\quad c_i, c_j, c_k = NO\}.\end{gathered}$$

Then, if there are 30 and 40 eligible items generated at A and B, 1200 candidate instances will be created for each of $\mathcal{P}$ and $\mathcal{P}'$. However, as the instance size rapidly increases with the pattern length (*i.e.,* the number of items required by a complex pattern), such a detection manner potentially leads to serious wastes of computational and storage resources [12], [15]. Even so, as early CEP systems processed only a few (≤ 10) complex patterns, the issue did not significantly impact their performance.

To make CEP systems scalable to the increasing number of complex patterns, a natural approach is to aggregate the common candidate instances across multiple complex patterns [15], [16], [21]–[23]. For example, the instances consisting of items at A and B can be aggregated across $\mathcal{P}$ and $\mathcal{P}'$, thus avoiding repeated creation. Multiple approaches [21]–[23] have been devised to minimize the overall instance size of given complex patterns through instance aggregation, where the key lies in constructing a *plan* that specifies which instances are aggregated across which complex patterns. Consider an example of jointly detecting four complex patterns $\mathcal{P}$ (in Fig. 2), $\mathcal{P}'$ (defined above), $\mathcal{P}''$ (requiring items at locations A, X, Y, and Z), and $\mathcal{P}'''$ (requiring items at D, X, Y, and Z), one ideal aggregation plan is aggregating the instances consisting of items at A and B across $\mathcal{P}$ and $\mathcal{P}'$, and those with items at X, Y, and Z across $\mathcal{P}''$ and $\mathcal{P}'''$.

For seeking the optimal aggregation plan, existing approaches go through the subsets of the data items required by given complex patterns for identifying aggregatable instances, and meanwhile utilize combinatorial optimization or heuristic methods to construct the plan [15], [16], [22]. However, for a complex pattern requiring l items (*i.e.,* with length l), it has $O(2^l)$ item subsets; this makes the solution space rapidly increase with the pattern length and number. Initially, the exponential solution space did not result in too much overhead, because existing approaches processed just dozens of complex patterns whose maximal lengths are $\sim$10.

With the development of mobile applications which keeps increasing the pattern length (more than 20 [16]) and number (up to thousands [15]), especially when they are equipped with generative SQL [24] (*i.e.,* complex patterns generated by large language models), the aggregation plan construction now becomes a new scalability bottleneck of CEP systems. For example, more than 10,000 seconds are required for constructing a plan for merely 100 complex patterns whose lengths are 20 [16]. Such a level of time costs makes aggregation plans have to be constructed offline, and incurs a considerable time gap for CEP systems as data streams are required to be processed on-the-fly. What is worse, due to the costs in construction, the quality of the plans constructed offline is still with tradeoffs, inducing long-term effects on the performance of CEP systems [15].

1.2 Contribution

In this paper, we explore how to make aggregation plan construction fast and scalable while maintaining high efficacy, so that it can well suffice the requirement of online detection, and can even run atop the emerging serverless platforms (involving complicated and distributed metalog-based synchronizations across stateless/stateful cloud functions). Our key idea is to utilize the latent *hierarchical cluster structure* among complex patterns. Concretely, we note that the complex patterns which share a lot of common candidate instances are similar with each other; if we link each complex pattern with its similar ones to build a graph, they will form closely linked clusters. Also, such clusters can be partitioned into multiple levels (*e.g.,* for a cluster of complex patterns which share the instances with items at A, B, and C, and another cluster sharing instances with items at A, B, and D, they can be partitioned from a higher-level cluster that shares the instances with items at A and B).

Delving deeper, we survey the mobile data streams considered in recent work (*e.g.,* bus GPS data [13], traffic flow data [16], smart home data [12], vehicle and sensor data [15], and city bike data [6]), finding that their items are mostly tagged with geographical locations. Over such data streams, a CEP system usually handles a number of widely distributed complex patterns (*e.g.,* covering a city), while every single complex pattern consists of the items within some small regions. As a result, the geographically close complex patterns oftentimes share the items at same locations; in contrast, the faraway ones exhibit little similarity. Such heterogeneous similarities then facilitate cluster partitioning. Thus, the hierarchical cluster structure is especially distinct among the complex patterns in mobile data streams.

Further dissecting the aggregation plan construction, as the complex patterns with many common instances are easy to form clusters, we can preferentially examine the intra-cluster ones to identify aggregatable instances, without the need to go through the exponential solution space. We fulfill the above idea by adapting the structural entropy minimization (SEM) algorithm onto a graph where complex patterns (as nodes) link their similar ones. The SEM algorithm provides structural entropy, a lightweight metric that measures the closeness among different levels of clusters purely based on the graph structure [25]. Specifically, we leverage the structural entropy to quickly locate the closely correlated complex patterns; if examining that there exist aggregatable instances among them, we then agglomerate them into clusters. We term the above guided construction method as *content-aware SEM*.

Moreover, for the "residual" instances that are not aggregated within clusters, we also devise a *differential compensation* mechanism to further identify the opportunities for aggregating them across clusters. Such a mechanism is realized via a cluster-level search, and thus is lightweight.

We conduct extensive experiments on real-world vehicle and sensor network data streams to evaluate our approach dubbed PatternInsight. The results demonstrate that PatternInsight can significantly accelerate instance aggregation. Compared to state-of-the-art approaches, it saves the aggregation plan construction time by 10× to 50×; also, such an advantage increases with the pattern length and number, indicating its high scalability. For the performances during detection, the instance size of PatternInsight is 40% lower than state-of-the-arts.

2 Preliminaries

2.1 Complex pattern

The complex patterns refer to user-defined combinations of certain data items (*a.k.a.,* primitive events). Each primitive event is represented by its occurrence time, event type, and a set of attributes. Concretely, the occurrence time is the timestamp when the corresponding data item is generated over data streams. The event type is specified by the essential contents in data items, which are specified by certain applications. In the example shown in Figs. 1 and 2, as traffic flow monitoring is strongly correlated with geographical locations, the primitive event types are specified as the locations where corresponding data items are generated, and the event attributes include speed, direction, accident and road construction information. Taking $\boldsymbol{Item}_i$ in Fig. 1 which is generated at the location A in Fig. 2 as an example, its primitive event type is specified as A; its occurrence time and event attributes are then respectively denoted by $A.t = t_i$ and $A.attr = \{v_i, d_i, a_i, c_i\}$.

Referring to [15], [16], [23], [26], we define each complex pattern $\mathcal{P}_n$ as a tuple $\mathcal{P}_n = \{E_n, T_n, R_n, O_n\}$. Here, E_n is the set of required primitive event types in a matched instance of $\mathcal{P}_n$. T_n is the maximal time interval, *i.e.,* the maximally allowed occurrence time difference between any pair of primitive events in a matched instance of $\mathcal{P}_n$. Then, R_n denotes the requirements on the attributes of primitive events. For example, let $\mathcal{P}_n = \mathcal{P}$ (in Fig. 2), it requires primitive events with types A, B, and C (*i.e.,* $E_n = \{A, B, C\}$); the maximally allowed occurrence time difference of $\mathcal{P}_n$ is 20 minutes (*i.e.,* $T_n = 20\,min$); then, R_n includes the requirements on vehicle speeds, directions, as well as accident and road construction information. Further, O_n refers to the *pattern operators*, which specifies the required logical relations among the primitive events. The four common pattern operators are **SEQ**, **AND**, **OR**, and **NOT** [15] [16].

1. A **SEQ** operator requires that: (1) there emerge all of the required primitive event types in an instance of the complex pattern; (2) the primitive event types appear in a required temporal order. For example, the complex pattern $\mathcal{P}$ in Fig. 2 requires that there emerge primitive events with the types being A, B, and C, and $A.t \le B.t \le C.t$; so the requirements on the logical relation and event types in $\mathcal{P}$ can be represented by $\mathbf{SEQ}\{A, B, C\}$. Also, for three primitive events which constitute a matched instance of $\mathcal{P}$, they should satisfy $C.t - A.t \le 20\,min$, as well as the requirements on $A.attr$, $B.attr$, and $C.attr$. Similarly, the requirements in complex pattern $\mathcal{P}'$ (in Section 1.1) can be represented by $\mathbf{SEQ}\{A, B, D\}$.

2. An **AND** operator in $\mathcal{P}_n$ requires the occurrence of all the required primitive event types during a time interval of length T_n, and has no requirement on their orders. For example, $\mathbf{AND}\{A, B, C\}$ can represent the requirements that there emerge abnormal vehicle speeds at locations A, B, and C, but monitors do not care about their orders.

3. An **OR** operator, as the name suggests, denotes a specified union of alternative primitive event types. Taking a complex pattern with $\mathbf{OR}\{A, B, C\}$ as an example, if a primitive event with any type in $\{A, B, C\}$ appears and its attributes satisfy R_n, such a primitive event is then a matched instance of $\mathbf{OR}\{A, B, C\}$.

4. A **NOT** operator requires the absence of an event type from some position in a complex pattern. We use $!E$ to denote the **NOT** operator on event type E. Consider a complex pattern P_n with $\mathbf{SEQ}\{A, B, !C, D\}$, it requires that there successively emerge slow speeds at locations A, B, and D within time T_n, but there is no primitive event with type C satisfying $A.t \le B.t \le C.t \le D.t$. In the example in Fig. 2, such a complex pattern can not only be used to detect phantom traffic jams, but also help locate the causes of jams — by excluding some possible locations.

In addition, we allow the *nested* logical requirements in complex patterns. That is, there are some complex patterns contain more than one logical operators, such as $\mathbf{SEQ}\{A, B, \mathbf{OR}\{C, D\}, E\}$ and $\mathbf{AND}\{\mathbf{SEQ}\{A, B\}, \mathbf{SEQ}\{C, D\}\}$.

2.2 Complex pattern detection

Complex pattern detection is performed by a Complex Event Processing (CEP) system [15], [17], [18], which continuously creates the combinations of input primitive events that serve as the candidate instances of complex patterns. For any complex pattern P_n, if there emerge matched instances that satisfy all the requirements in $\{E_n, T_n, C_n, O_n\}$, the CEP systems then return the matched instances.

Early CEP systems adopt the non-deterministic finite automaton (NFA) mechanism [15], [17], [18], which independently creates and stores the candidate instances for each complex pattern when a new primitive event is generated. Taking the complex pattern with $\mathbf{SEQ}\{A, B, C, D\}$ as an example. When a primitive event with type A arrives, NFA creates a candidate instance with this event. Then, if there are 3 events with type A and 2 events with type B, NFA creates $3 \times 2 = 6$ candidate instances for $\mathbf{SEQ}\{A, B, C, D\}$, and so on. Once there emerges an instance containing the primitive events with types A, B, C, and D and satisfying the requirements on time interval and attributes, NFA returns such instance and reports detecting the complex pattern with $\mathbf{SEQ}\{A, B, C, D\}$.

The NFA mechanism induces the worst-case exponential size of the candidate instances of each complex pattern in terms of the pattern length. Here, the pattern length refers to the number of primitive events that a complex pattern requires (*e.g.,* for a complex pattern with $\mathbf{SEQ}\{A, B, C, D\}$, its pattern length is 4). However, since current CEP systems need to parallelly process hundreds to thousands of complex patterns, the candidate instance size becomes the main bottleneck of detection efficiency. This is because, during detection process, the candidate instances will occupy memory and consume processing time (for creating them and checking if they match any complex pattern) of CEP systems; a huge number of instances can negatively impact the performance of CEP systems in terms of detection latency, throughput, and so on.

To address this, the instance aggregation approach is widely adopted to reduce instance size through aggregating the instances of common sub-patterns across multiple complex patterns [15], [16], [21]–[23]. For a pattern $\mathcal{SP}$, we say it is a sub-pattern of another pattern $\mathcal{P}$ if: (1) the required primitive event types of $\mathcal{SP}$ is a subset of those required by $\mathcal{P}$; (2) the requirements on logical relation and event attributes of $\mathcal{SP}$ are compatible with the requirements of $\mathcal{P}$; (3) the allowed time interval of $\mathcal{SP}$ is not smaller than that of $\mathcal{P}$. For example, consider a pattern $\mathcal{SP}$ with $\mathbf{SEQ}\{A, B\}$ which describes that the west-to-east speeds at locations A and B (in Fig. 2) successively drop below $40\,km/h$ within 20 minutes and there is no accident or construction, SP is a common sub-pattern shared by $\mathcal{P}$ and $\mathcal{P}'$ (in Section 1).

By aggregating the candidate instances of $\mathcal{SP}$ across $\mathcal{P}$ and $\mathcal{P}'$, the CEP system can avoid repeatedly creating and storing such instances when jointly detecting $\mathcal{P}$ and $\mathcal{P}'$. More specifically, with instance aggregation, the CEP system will not start creating the instances of $\mathcal{P}$ and $\mathcal{P}'$ until there emerge matched instances of SP, instead of independently creating the instances of $\mathcal{P}$ and $\mathcal{P}'$ from scracth. As detecting the sub-pattern $\mathcal{SP}$ is a precondition of detecting $\mathcal{P}$ and $\mathcal{P}'$, the instance aggregation approach thus can markedly save the computational and storage resources when waiting for the matched instances of $\mathcal{SP}$. Thus, in current mobile applications that need to jointly detect a number of complex patterns, there are tremendous chances for improving detection efficiency via instance aggregation.

For implementing instance aggregation, the key is to construct an aggregation plan among complex patterns.

Definition 1. *(***Aggregation plan***). Given multiple complex patterns that need to be parallelly detected, the aggregation plan (denoted by* $\mathbf{AP}$*) specifies the instances of which sub-patterns are aggregated across which complex patterns.*

The aggregation plan usually follows a tree-like structure, as shown in Fig. 3; each internal vertex is marked by a sub-pattern shared by its child vertices. Consider four complex patterns $\{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4\}$ respectively with $\mathbf{SEQ}\{A, B, C\}$, $\mathbf{SEQ}\{A, B, D\}$, $\mathbf{SEQ}\{A, X, Y, Z\}$ and $\mathbf{SEQ}\{D, X, Y, Z\}$, there are multiple optional aggregation plans across them. One is aggregating the instances of $\mathbf{SEQ}\{A, B\}$ across $\mathcal{P}_1$ and $\mathcal{P}_2$, and aggregating the instances of $\mathbf{SEQ}\{X, Y, Z\}$ across $\mathcal{P}_3$ and $\mathcal{P}_4$; another is aggregating the primitive events (instances) with type A across $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$. Obviously, different aggregation plans will induce different instance sizes and the resulting computational overhead during detection.

2.3 Problem statement

In this paper, we consider the instance aggregation approach for realizing efficient complex pattern detection over data streams. Our target is, given a complex pattern workload $\mathcal{W} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N\}$ that need to be parallelly detected over an input data stream of CEP system, constructing an aggregation plan that can help minimize the expected overall candidate instance size, and then implementing instance aggregation based on such a plan during real-time complex pattern detection.

For constructing the aggregation plan, prior arts primarily focus on a statistically directed manner. That is, they traverse the sub-patterns of given complex patterns, and meanwhile for each sub-pattern, determine if its instances are aggregatable (*i.e.,* can be aggregated across at least two complex patterns), and estimate the instance size reduction after aggregation based on the stream statistics [16], [23], [26]. Here, the stream statistics refer to the frequencies of each primitive event type and the probabilities of event attributes satisfying the required conditions. Based on stream statistics, prior arts first estimate the expected instance size of each sub-pattern during its allowed time interval; then, together with the number of complex patterns sharing such a sub-pattern, they give the estimated instance size reduction after aggregating the instances of this sub-pattern. Further, the sub-patterns with aggregatable instances, as well as their estimated instance size reductions, are fed into customized combinatorial optimization or heuristic algorithms for constructing the aggregation plan.

However, such construction methods induce an exponential solution space and exhibit low-scalability, since the number of sub-patterns of each complex pattern exponentially increases with its pattern length. As introduced in Section 2.2, sub-patterns of a complex pattern are the combinations of its event type subsets and compatible requirements on logic relations and attributes, and there are $O(2^l)$ event type subsets for a complex pattern with length l. With the development and popularization of mobile applications, the CEP systems over mobile data streams now need to handle up to thousands of complex patterns whose maximum lengths are already more than 20 [6], [16]. This trend leads to prohibitive complexities of current construction methods, making aggregation plans have to be constructed offline. That is, upon given the complex pattern workload $\mathcal{W}$, the CEP system needs to spend considerable time on constructing the aggregation plan, before executing instance aggregation during real-time detection.

As a result, the longer the time that aggregation plan construction requires, there is a longer time gap for CEP systems as data streams need to be processed in real time. Also, the prohibitive complexity will further hinder timely updates of aggregation plans, when users define and add new

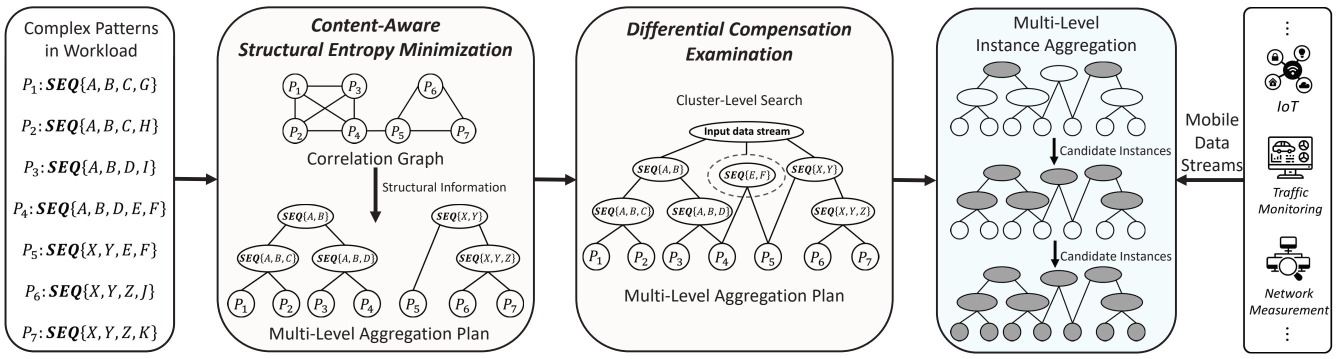


Fig. 3: Architectural overview of PatternInsight.

complex patterns into the workload. Moreover, the required stream statistics of some large-scale, newly-generated data streams are hard to obtain [15], [27]. These result in the scalability bottleneck in aggregation plan construction.

What is worse, due to the scalability issue in construction, even the quality of the plans constructed by prior arts offline is still with tradeoffs. The most significant limitation is that prior arts are primarily constrained on the single-level instance aggregation. That is, the aggregation plan just enables aggregating common instances when detecting complex patterns; since detecting sub-patterns is the precondition of detecting complex patterns, the detection efficiency can be further improved if aggregating the instances of the sub-patterns of sub-patterns and so on (*i.e.,* multi-level instance aggregation). However, when considering multi-level instance aggregation, the solution space of aggregation plan construction will be enlarged by times, further exacerbating the scalability issue of prior arts.

3 Design of PatternInsight

In this paper, we present PatternInsight, a novel instance aggregation framework that makes the (multi-level) aggregation plan construction fast and scalable, while maintaining high efficacy, so that it can well support online complex pattern detection. The key idea of PatternInsight is to leverage the *latent hierarchical* cluster structure among complex patterns to quickly construct the aggregation plan. Specifically, for a group of complex patterns where a lot of common candidate instances can be aggregated across, they are similar to each other (in terms of the instance similarities); if linking each complex pattern (as a node) with its similar ones to build a graph (termed correlation graph of complex patterns), such a group of complex patterns will form closely linked clusters. Based on this, we can speed up aggregation plan construction by preferentially examining the common sub-patterns of intra-cluster complex patterns to identify aggregatable candidate instances, without the need to go through the exponential solution space. Fig. 3 presents the mian components of PatternInsight.

(1) *Content-Aware Structural Entropy Minimization (CASEM)*. To implement the above idea, we need to quickly locate the closely linked nodes over the correlation graph. We achieve this based on the widely recognized principle: through properly partitioning similar or correlated elements in a given system into groups, the uncertainty (entropy) embedded into the system can be reduced. The reduced entropy is taken as the amount of the information decoded from the given system via such a partition [25], [28]. This is also the basis of utilizing hierarchical code or decision tree to reduce the expected code length (*i.e.,* Shannon entropy).

By this, we take the hierarchical cluster structure among complex patterns as the information that we intend to decode from the correlation graph. We thus locate the closely linked nodes by determining that partitioning which nodes into a group (cluster) can help minimize the entropy embedded into the structure of correlation graph (termed structural entropy [25]). The structural entropy reduction after partitioning is computed based on local graph structures, so that it can serve as a lightweight metric of node closeness.

With the above insight, CASEM maps the aggregation plan construction into a “clustering” process. The complex patterns which share a common sub-pattern are agglomerated into a cluster, and such clusters can be further agglomerated into higher-level ones. For example, the two clusters of complex patterns, which respectively share the sub-patterns with $\mathbf{SEQ}\{A, B, C\}$ and $\mathbf{SEQ}\{A, B, D\}$, can be agglomerated into a higher-level cluster sharing the sub-pattern with $\mathbf{SEQ}\{A, B\}$. Different from original procedures of structural entropy-based clustering [25] that purely depend on the graph structure, CASEM takes structural closeness as the guidance to determine which complex patterns are preferentially examined for identifying aggregatable instances; if there exist aggregatable instances among the examined complex patterns, CASEM then agglomerates them into a cluster. Taking the complex pattern represented by each node as its content, we name the above method as content-aware structural entropy minimization.

(2) *Differential Compensation Examination (DCE)*. CASEM allows the instance aggregation within each cluster, but is there any chance for further reducing instance size via inter-cluster instance aggregation? To answer this question, DCE performs a lightweight cluster-level search for aggregating the “residual” instances that are not aggregated within clusters. Such “residual” instances are usually of the *differential* parts of patterns (*e.g.,* for the pattern requiring $\mathbf{SEQ}\{X, Y, E, F\}$ in Fig. 3 that belongs to the cluster sharing $\mathbf{SEQ}\{X, Y\}$, its differential part is with $\mathbf{SEQ}\{E, F\}$).

The aggregation plan produced by CASEM (together with DCE) is tree-like, where each leaf vertex [1] represents

1. Throughout the paper, vertices refer to the elements in tree-like aggregation plan, and nodes refer to the elements in graph.

a complex pattern; each internal vertex represents a set of complex patterns sharing a same sub-pattern, and is marked by such a sub-pattern (as shown in Fig. 3); a higher-level vertex is marked by a sub-pattern shared by its child vertices. Based on this, PatternInsight can achieve efficient online detection by fully leveraging the tree-like multi-level aggregation plan to create instances in a top-to-down manner. That is, over input data streams, the candidate instances of a vertex will not be created until there emerge matched instances of its all parent vertices. By this, PatternInsight can avoid repeatedly creating the candidate instances of sub-patterns at all levels, thereby significantly improving the detection efficiency. When there emerge matched instances of any leaf vertex, the CEP system reports that the corresponding complex pattern is detected.

4 Content-Aware Structural Entropy Minimization

Given the complex patterns in the workload $\mathcal{W}$ that need to be parallelly detected over input data streams, PatternInsight aims at quickly constructing an aggregation plan for them while maintaining high efficacy. To achieve this, CASEM is designed based on the insight: the complex patterns sharing a number of common candidate instances will form clusters over the correlation graph where each complex pattern links its similar ones; through preferentially examining the intra-cluster complex patterns, the aggregatable instances can be quickly identified.

The preliminary step of implementing above idea is efficiently uncovering the hierarchical cluster structure from the correlation graph of complex patterns, which serves as the guidance to preferential examination. To this end, CASEM utilizes the structural information theory, and maps aggregation plan construction into a "clustering" process that minimizes the structural entropy embedded into correlation graph. Before presenting the designs of CASEM, we give an introduction to the structural information theory.

4.1 Structural Information Theory Basics

On the basis of classical Shannon entropy, the structural entropy is proposed by Li *et al.* in [25] to measure the uncertainty embedded into graph structure. Specifically, let $G = (V, E)$ denote a graph, where V and E respectively refer to the node and edge set. Consider a random walk process over G that a node diffuses messages to its one randomly chosen neighbor. Let d_n denote the size of the neighbors of a node $v_n \in V$ (*i.e.,* the degree of v_n), the probability of v_n being chosen as the message receiver is $\frac{d_n}{2|E|}$, where $2|E|$ is the sum of the degrees of all nodes in G. With such a probability distribution of message receiver, according to the Shannon information theory, the minimum expected code length for encoding the message receiver is

$$H(G) = -\sum_{v_n \in V} \frac{d_n}{2|E|} \log_2 \frac{d_n}{2|E|}; \tag{1}$$

$H(G)$ is also the Shannon entropy that measures the uncertainty of the message receiver during the random walk over graph G. In structural information theory, $H(G)$ is treated as the original entropy embedded into the structure of G.

In order to reduce the structural entropy, as well as the expected code length of message receiver, the structural information theory proposes to encode nodes via partitioning them into multi-level clusters, in a manner similar to the hierarchical encoding. Concretely, by partitioning nodes into multiple clusters, the codeword of each node then consists of: (1) the codeword of the cluster it belongs to, and (2) its codeword within the cluster. Then, in each step of the random walk, if the receiver belongs to the same cluster of the chosen node, the codeword of cluster can be omitted. Thus, through partitioning structurally close nodes into clusters, the expected code length of receivers can be reduced. As such, the expected code length can be minimized through seeking an optimal hierarchy that partitions nodes into multi-level clusters. The multi-level cluster hierarchy is represented by a partition tree [25].

Definition 2. (**Partition tree of graph** G). *We let T denote the partition tree for the node clustering of $G = (V, E)$. Specifically,*

(1) the root vertex η of T represents the universal node set V;

(2) each leaf vertex γ in T represents a node v_n in V;

(3) each internal vertex $\alpha \in T$ represents a node cluster in G and contains the nodes belonging to such a cluster. Let $\{\beta_1, \beta_2, \ldots, \beta_s\}$ denote the child vertices of α, the nodes respectively belonging to $\beta_1, \beta_2, \ldots,$ and β_s form a disjoint partition of the nodes in their parent vertex α.

With the partition tree, nodes can be encoded in a level-by-level manner. We denote the expected code length for encoding a node with the partition tree T by $H^T(G)$, which is defined as the *structural entropy* [25] of graph G with T.

Definition 3. (**Structural entropy**). *Given an undirected graph G, the structural entropy with partition tree T is*

$$H^T(G) = -\sum_{\xi \in T, \xi \neq \eta} \frac{l_\xi}{E(G)} \cdot \log_2 \frac{E(\xi)}{E(\xi^-)}. \tag{2}$$

Here, l_ξ is the number of edges linking one node in vertex (cluster) ξ and one out of ξ. $E(G)$ denotes the volume of graph G, which is the sum of the degrees of nodes in G, and $E(\xi)$ denotes the volume of nodes belonging to vertex (cluster) ξ; specially, for each leaf vertex, its volume is equal to corresponding node degree. ξ^- represents the parent vertex of ξ in partition tree T.

Eqn.(2) presents a multi-level encoding manner of nodes in G. We take a partition tree with the depth being 3 as an example for explaining Eqn. (2). Specifically, for a leaf vertex γ representing the node v_n, let $(\gamma^-)^-$ denote its parent's parent vertex. Given the graph volume $E(G) = 2|E|$, the probability that the message receiver belongs to $(\gamma^-)^-$ is $\frac{E[(\gamma^-)^-]}{E(G)}$ (with the entropy being $-\log_2 \frac{E[(\gamma^-)^-]}{E(G)}$); then, given message receiver in $(\gamma^-)^-$, the entropy of message receiver belonging to γ^- is $-\log_2 \frac{E(\gamma^-)}{E[(\gamma^-)^-]}$; last, the entropy of message receiver being γ is $-\log_2 \frac{d_n}{E(\gamma^-)}$. Moreover, the probability $\frac{l_\xi}{E(G)}$ characterizes that, only if a node out of ξ diffuses a message to a node within ξ, the codeword of the vertex ξ is then required for encoding the message receiver.

Further, through partitioning structurally close nodes into clusters and thereby improving the chance of intra-cluster walk, the expected code length for encoding message receivers (*i.e.,* structural entropy) can be reduced. Accordingly, through minimizing the structural entropy, we can seek a hierarchy partition tree which uncovers the multi-level cluster structure among nodes.

4.2 Multi-Level Aggregation Plan Construction

With Definition 2, it can be observed that the partition tree of graph shares a similar topology with the multi-level aggregation plan that we aim to construct. Concretely, each leaf vertex in aggregation plan represents a complex pattern, while in partition tree, represents a node in graph. Then, an internal vertex in aggregation plan consists of a set of complex patterns sharing a common sub-pattern and is marked by such a sub-pattern (as shown in Fig. 3), and each internal vertex in partition tree represents a set of nodes belonging to a same cluster. Then, the the root vertex represents the universal set of nodes (complex patterns).

Based on above observations, we blend the two construction processes of the aggregation plan and the partition tree of the correlation graph of complex patterns.

Definition 4. *(***Correlation graph of complex patterns***.) Over the correlation graph $G = (V, E)$, each complex pattern (represented by a node) links its similar ones.*

The similarities among complex patterns can be quickly computed by counting their common required primitive event types; this is because, for any pair of complex patterns, requiring common primitive event types is a necessary condition for the existence of common sub-patterns between them. Then, we build the correlation graph by linking each complex pattern with its similar ones. Over the correlation graph, the complex patterns (nodes) which share common sub-patterns will form closely linked clusters. By uncovering and preferentially examining the intra-cluster complex patterns, PatternInsight can thus quickly identify aggregatable instances.

Initially, both the aggregation plan $\mathbf{AP}$ and the partition tree T are one-level; that is, the leaf vertices directly connect with the root (as shown in the left part of Fig. 4). Then, the construction process is implemented by creating and agglomerating leaf vertices into (multi-level) internal vertices. To this end, there are two types of operations (*i.e.*, combining and merging) in aggregation plan construction.

4.2.1 *Combining operation in aggregation plan construction*

As shown in Fig. 4, for any pair of sibling vertices α and β, the combining operation creates an internal vertex ξ as their parent vertex (*i.e.*, $\alpha^- = \beta^- = \xi$); the vertex ξ then represents the union node set of the nodes (sets) represented by α and β (*i.e.*, $\xi = \alpha \cup \beta$). In partition tree construction, such a combining operation represents agglomerating the nodes (or clusters) represented by vertices α and β into a (higher-level) cluster ξ.

As the combining operation updates the partition tree of correlation graph, it will change the structural entropy accordingly. Let T and $T^{C(\alpha,\beta)}$ respectively denote the partitioning trees before and after the combining operation on vertices α and β, the structural entropy change induced by the combining operation is

$$\Delta H^T(G, C(\alpha,\beta)) = H^T(G) - H^{T^{C(\alpha,\beta)}}(G). \qquad (3)$$

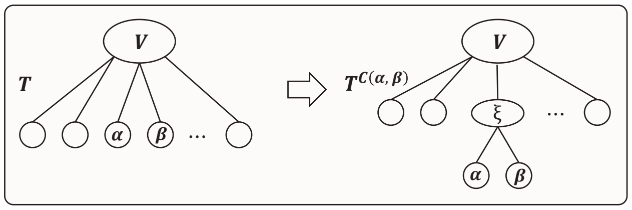


Fig. 4: An example of the *combining* operation in partition tree construction.

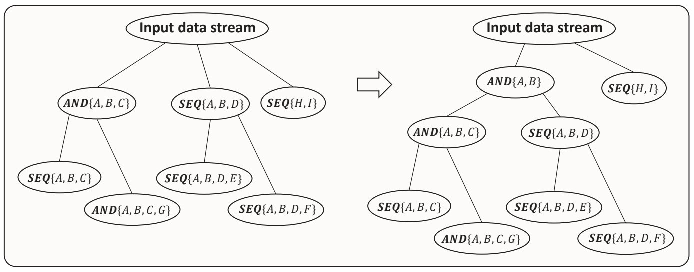


Fig. 5: An example of the *combining* operation in aggregation plan construction, in which we create a higher-level vertex marked by $\mathbf{AND}\{A, B\}$ for a pair of vertices respectively marked by $\mathbf{AND}\{A, B, C\}$ and $\mathbf{SEQ}\{A, B, D\}$.

Taking Eqn. (2) into Eqn. (3), we have

$$\begin{aligned}
&\Delta H^T(G, C(\alpha,\beta)) \\
=& \underbrace{-\left(\frac{l_\alpha}{E(G)}\log_2\frac{E(\alpha)}{E(\alpha^-)} + \frac{l_\beta}{E(G)}\log_2\frac{E(\beta)}{E(\beta^-)}\right)}_{\text{computed over } T} \\
&- \underbrace{\left\{-\left(\frac{l_\alpha}{E(G)}\log_2\frac{E(\alpha)}{E(\xi)} + \frac{l_\beta}{E(G)}\log_2\frac{E(\beta)}{E(\xi)}\right)\right\}}_{\text{computed over } T^{C(\alpha,\beta)}} \\
&- \underbrace{\left(-\frac{l_\xi}{E(G)}\log_2\frac{E(\xi)}{E(\xi^-)}\right)}_{\text{computed over } T^{C(\alpha,\beta)}}.
\end{aligned} \qquad (4)$$

As presented in Eqn. (4), the structural entropy change resulted by the combining operation in Fig. 4 only regards to the vertices α and β in T (the first line in Eqn. (4)), as well as α, β, and ξ in $T^{C(\alpha,\beta)}$ (the second and third lines). The computation method for any term in Eqn. (4) (*e.g.*, $\frac{l_\alpha}{E(G)}\log_2\frac{E(\alpha)}{E(\xi)}$) is given in Definition 2, which can be computed by counting the edges of nodes belonging to α, β, and ξ. Thus, the structural entropy change can be locally computed, and serves as a lightweight metric of the closeness among nodes over graph. If $\Delta H^T(G, C(\alpha,\beta)) > 0$, it indicates that combining α and β into a (higher-level) cluster can improve the chance of intra-cluster random walk over the graph, and thus can reduce the structural entropy.

In original structural entropy-based clustering [25], it greedily combines the vertex (cluster) pairs whose combination brings more entropy reduction, in order to seek the partition tree with the minimum structural entropy (in polynomial time). In this paper, our solution for aggregation plan construction is the content-aware structural entropy minimization, where the entropy reduction after combination is taken as the guidance to determine which vertices are preferentially examined for identifying common sub-pattern. This is because for a pair of vertices, if their combination can induce a high entropy reduction, the nodes in the two vertices form a closely linked cluster. Then, over the correlation graph, as each complex pattern (as a node) links its similar ones, the closely linked similar complex patterns are more probably to have aggregatable instances.

Concretely, in aggregation plan construction, given current aggregation plan $\mathbf{AP}$, we first compute the structural entropy changes after combining the nodes (or node sets) represented by each sibling vertex pair into a (higher-level) cluster with Eqn. (4). Then, we preferentially examine the sibling vertex pairs with higher entropy reduction for identifying the common sub-pattern. If there exists a common sub-pattern between the examined vertex pair, we then create a parent vertex for them and mark the newly created vertex by their common sub-pattern; the two sets of complex patterns in the vertex pair are agglomerated into their parent vertex. Fig. 5 presents an example of the combining operation.

The rules for common sub-pattern identification will be introduced later on.

4.2.2 Merging operation in aggregation plan construction

As shown in Fig. 6, for a pair of sibling vertices α and β, if one or two of them are internal vertices, the merging operation is optional on them in partition tree construction. Different from combining operation, the merging operation directly merges the nodes in α and β together, without creating a parent vertex for them.

Let T and $T^{M(\alpha,\beta)}$ respectively denote the partition trees before and after the merging operation on α and β, the resulted structural entropy change is

$$\Delta H^{T}(G, M(\alpha,\beta)) = H^{T}(G) - H^{T^{M(\alpha,\beta)}}(G). \tag{5}$$

Similar to Eqns. (3) and (4), by taking Eqn. (2) into Eqn. (5), we obtain

$$\begin{aligned}
&\Delta H^{T}(G, M(\alpha,\beta)) = \\
&\underbrace{-\left(\sum_{\xi\in\{\alpha,\beta\}} \frac{l_\xi}{E(G)}\log_2\frac{E(\xi)}{E(\xi^-)} + \sum_{\xi\,\middle|\,\xi^-=\alpha,\beta} \frac{l_\xi}{E(G)}\log_2\frac{E(\xi)}{E(\xi^-)}\right)}_{\text{computed over } T} \\
&-\underbrace{\left\{-\left(\frac{l_\alpha}{E(G)}\log_2\frac{E(\alpha)}{E(\alpha^-)} + \sum_{\xi\,\middle|\,\xi^-=\alpha} \frac{l_\xi}{E(G)}\log_2\frac{E(\xi)}{E(\alpha)}\right)\right\}}_{\text{computed over } T^{M(\alpha,\beta)}}.
\end{aligned} \tag{6}$$

In Eqn. (6), the structural entropy change resulted by the merging operation in Fig. 7 only regards to α, β, and their child vertices in T (the first line in Eqn. (6)), as well as α and the child vertices of α in $T^{M(\alpha,\beta)}$ (the second line). Thus, Eqn. (6) can also be locally and efficiently computed. If $\Delta H^{T}(G, M(\alpha,\beta)) > 0$, the merging operation on vertices α and β can help reduce the structural entropy. Both of the structural entropy reductions of the combining and merging operations serve as the guidance to locate closely correlated nodes (complex patterns).

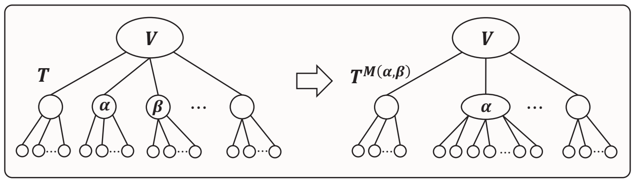


Fig. 6: An example of the *merging* operation in partition tree construction.

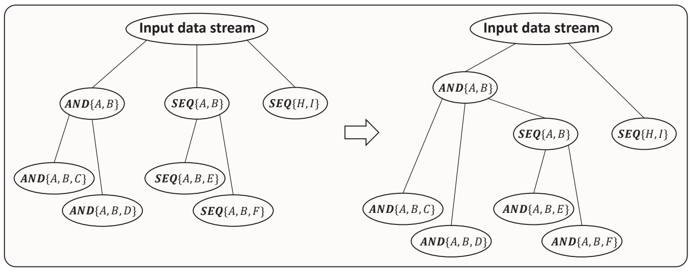


Fig. 7: An example of the *merging* operation in aggregation plan construction.

When updating the aggregation plan, PatternInsight usually adopts the combining operation in aggregation plan construction (as shown in Fig. 5), where the newly created parent vertex is marked by the identified common sub-pattern. Specially, for a pair of examined vertices (*e.g.,* α and β), if the marking pattern of a vertex (*e.g.,* α) is exactly the sub-pattern of another (*e.g.,* β), the identified common sub-pattern is the marking pattern of α. In such cases, we can adopt the merging operation. Fig. 7 gives an example of the merging operation in aggregation plan construction. As the instances which match $\mathbf{SEQ}\{A,B\}$ also match $\mathbf{AND}\{A,B\}$, $\mathbf{AND}\{A,B\}$ is identified as the common sub-pattern of such two vertices. Then, the complex patterns in the vertex $\mathbf{SEQ}\{A,B\}$ are merged into the vertex $\mathbf{AND}\{A,B\}$; the vertex $\mathbf{SEQ}\{A,B\}$ then becomes a child vertex of $\mathbf{AND}\{A,B\}$. By this, when jointly detecting $\mathbf{SEQ}\{A,B\}$ and $\mathbf{AND}\{A,B\}$, PatternInsight can avoid repeatedly creating and storing their common instances.

4.2.3 Common sub-pattern identification

The entropy reductions induced by combination or merging operations serve as the guidance to preferentially examining which vertex pairs to identify common sub-patterns. Then, in each examination, the common sub-pattern are identified with the following rules.

Recall Section 2.1, the complex patterns are defined with the requirements of primitive event types, logic relations, time intervals, and event attributes. Then, for a pair of complex patterns, one necessary condition of the existence of a common sub-pattern is that they have common subsets of primitive event types. Based on this preliminary condition, we further elaborate the rules for common sub-pattern identification with the requirements of logic relations, time intervals, and event attributes.

For the logic requirements defined by the operators $\mathbf{AND}$, $\mathbf{SEQ}$, $\mathbf{OR}$, and $\mathbf{NOT}$, we give the sub-pattern identification rules as below:

- For a pair of patterns both with $\mathbf{AND}$, the identified common sub-pattern is the maximal common subset of the two event type sets. The example is, given two patterns respectively with $\mathbf{AND}\{A, B, C\}$ and $\mathbf{AND}\{A, B, D\}$, the identified common sub-pattern is with $\mathbf{AND}\{A, B\}$.
- For a pair of patterns both with $\mathbf{SEQ}$, we identify the common sub-pattern as the maximal common sequence of primitive event types. For example, given $\mathbf{SEQ}\{A, B, C, D\}$ and $\mathbf{SEQ}\{B, C, E\}$, identifying $\mathbf{SEQ}(B, C)$ as their common sub-pattern.
- For two patterns respectively with $\mathbf{AND}$ and $\mathbf{SEQ}$, we also extract the the maximal common subset of primitive event types. The example is that the instances matching $\mathbf{SEQ}\{B, C, E\}$ also match $\mathbf{AND}\{B, C, E\}$.
- For the event types associated with the operator $\mathbf{OR}$, we identify sub-patterns with the union of the event types in $\mathbf{OR}$, such as given $\mathbf{SEQ}\{B, C, E\}$ and $\mathbf{AND}\{C, D, \mathbf{OR}\{E, F\}\}$, the identified sub-pattern is $\mathbf{AND}\{C, \mathbf{OR}\{E, F\}\}$. Then, we take each event type associated with operator $\mathbf{NOT}$ as a separate type, such as for $\mathbf{SEQ}\{A, !B, C, D\}$ and $\mathbf{SEQ}\{A, !B, C, E\}$, identifying the common sub-pattern $\mathbf{SEQ}\{A, !B, C\}$.

The rules for the patterns with nested operators are given in Appendix A in the supplemental material. Further, for a sub-pattern identified from a pair of patterns, we set the condition on time interval as the maximum allowed interval of the two patterns; then, we set the attribute requirements in such a sub-pattern via an or operation (*i.e.,* $\vee$) on the requirements in the two patterns. This is because, for an instance, as long as it satisfies the requirements of one pattern, it should be taken as the matched instance of the identified common sub-pattern.

4.2.4 CASEM algorithm

With the combining and merging operations introduced before, as well as the rules for common sub-pattern identification, we now give the CASEM algorithm for multi-level aggregation plan construction.

As shown in Algorithm 1, the input of the CASEM algorithm is the correlation graph G. Initially, the aggregation plan $\mathbf{AP}$ is one-level. Then, we add the sibling vertex pairs into the candidate list CL, and determine which of them are preferentially examined based on the entropy reductions of them. As $\mathbf{AP}$ is taken as a partition tree of the correlation graph G, the entropy reductions induced by the combining and merging operations (*i.e.,* $\Delta H^{\mathbf{AP}}(G, C(\alpha, \beta))$ and $\Delta H^{\mathbf{AP}}(G, M(\alpha, \beta))$) are computed with Eqns. (4) and (6). Notably, at the beginning, since all the nodes (complex patterns) are leaf vertices, we only need to compute the entropy reductions of combining operations.

Based on the entropy reductions of the vertex pairs in CL, we preferentially examine the one with the maximum entropy reduction (denoted by (α, β)); if there exists the common sub-pattern of α and β, we conduct the merging or combining operation on them. Then, we update the candidate list CL; when conducting the combining operation (*e.g.,* combining α and β into ξ), we remove the vertex pairs which contain α or β from CL, and add the sibling vertex pairs having ξ into CL; when conducting merging operation (*e.g.,* merging β into α), we first remove vertex pairs having β from CL, and then add β together with its new siblings

Algorithm 1: CASEM algorithm.

```
   Input: Complex pattern correlation graph $G$;
   Output: Multi-level aggregation plan $\mathbf{AP}$;
1  Initialize aggregation plan $\mathbf{AP}$;
2  Add sibling vertex pairs into candidate list $CL$;
3  do
4      for each vertex pair $(\alpha, \beta)$ newly added into $CL$ do
5          entropy reduction $\Delta H(\alpha, \beta) \leftarrow$
             $\max\{\Delta H^{\mathbf{AP}}(G, C(\alpha, \beta)), \Delta H^{\mathbf{AP}}(G, M(\alpha, \beta))\}$;
6      end
7      Examine the vertex pair $(\alpha, \beta)$ with maximum
         entropy reduction;
8      if existing common sub-pattern then
9          Merge or combine $\alpha, \beta$ and update $\mathbf{AP}$;
10         Update candidate list $CL$;
11     else
12         Remove $(\alpha, \beta)$ from $CL$;
13     end
14 while there is vertex pair $(\alpha, \beta)$ in $CL$ with
     $\Delta H^{\mathbf{AP}}(G, C(\alpha, \beta)) > 0$ or $\Delta H^{\mathbf{AP}}(G, M(\alpha, \beta)) > 0$;
15 return aggregation plan $\mathbf{AP}$.
```

into CL. In each iteration, we only need to compute the entropy reductions of the vertex pairs newly added into CL.

In summary, the CASEM algorithm mainly consists of three steps: (1) computing the entropy reductions of vertex pairs in candidate list CL, which can be executed in parallel; (2) examining the vertex pair with the maximum entropy reduction; (3) updating $\mathbf{AP}$ and CL, and returning to step (1). Referring to Eqns. (4) and (6), the entropy reduction of any vertex pair, and the updates of $\mathbf{AP}$ and CL all can be locally computed. The CASEM algorithm stops when there is no combining or merging operation can reduce the structural entropy. Notably, CASEM can support the online cases with limited time for aggregation plan construction. This is because Algorithm 1 optimizes the aggregation plan in each iteration on the basis of that from last iteration; when the deadline of returning aggregation plan arrives, Algorithm 1 can terminate examination and return the current-best aggregation plan.

5 DIFFERENTIAL COMPENSATION EXAMINATION

5.1 DCE algorithm

The CASEM algorithm focuses on preferentially examining the intra-cluster complex patterns for identifying common sub-patterns (*i.e.,* aggregatable instances). After completing CASEM, if the deadline for returning the aggregation plan has not arrived, we proceed to conduct the Differential Compensation Examination (DCE) mechanism to further optimize the aggregation plan. DCE mainly aims to identify the opportunities for inter-cluster instance aggregation, which serves as compensation for the focus of CASEM on intra-cluster aggregation.

In the aggregation plan returned by CASEM, the complex patterns in a same cluster (*i.e.,* belonging to a same vertex) shares the marking sub-pattern of their parent vertex; the primitive event types required by such a sub-pattern

Algorithm 2: DCE algorithm.

```
Input: Complex pattern correlation graph $G$,
       Aggregation plan $\mathbf{AP}$ returned by CASEM;
Output: Multi-level aggregation plan $\mathbf{AP}$;
1  Add cluster pairs with inter-cluster edges into
     cluster-level candidate list $CCL$;
2  for each cluster pair $(\xi, \eta)$ in $CCL$ do
3      similarity $s(\xi, \eta) \leftarrow$ the number of inter-cluster
         edges between $\xi$ and $\eta$;
4  end
5  while $CCL$ is not empty do
6      $(\xi, \eta) \leftarrow$ the cluster pair in $CCL$ with the highest
         similarity;
7      for each child vertex pair $(\alpha, \beta)$ of $\xi$ and $\eta$ with
         inter-cluster edges do
8          if existing common sub-pattern between the
             differential parts of $\alpha$ and $\beta$ then
9              Create a new parent vertex for $(\alpha, \beta)$ in
                 $\mathbf{AP}$;
10         end
11     end
12     Remove $(\xi, \eta)$ from $CCL$;
13 end
14 return aggregation plan $\mathbf{AP}$.
```

(vertex) are a common subset of those required by its child vertices. Thus, the aggregation plan returned by CASEM leaves the residual instances of the differential parts of vertices. For a complex pattern $\mathcal{P}_n$, E_n denotes the set of its required primitive event types; let E_n^- denote the set of primitive event types required by its parent vertex, the differential part of P_n consists of the event types in the residual set $E_n \backslash E_n^-$, together with the requirements on logic, time and attributes for such event types. Similarly, the marking sub-patterns of internal vertices also have differential parts.

As the residual instances are not aggregated within clusters, DCE seeks opportunities for aggregating them across clusters and is implemented via a cluster-level search. Specifically, as shown in Algorithm 2, for any pair of clusters in $\mathbf{AP}$, if there exist the inter-cluster edges between them, DCE adds them into the cluster-level candidate list CCL. Here, the inter-cluster edge refers that its two endpoint nodes respectively belong to two different clusters. For a pair of clusters, having inter-cluster edges is a necessary condition of having aggregatable residual instances.

DCE then preferentially examines the cluster pairs with higher *similarities*. Specifically, for each cluster pair $\big(\xi = \{\alpha_1, \ldots, \alpha_m\}, \eta = \{\beta_1, \ldots, \beta_n\}\big)$ in CCL, where $\{\alpha_1, \ldots, \alpha_m\}$ and $\{\beta_1, \ldots, \beta_n\}$ respectively denote the complex patterns (nodes) belonging to ξ and η, we measure their similarity $s(\xi, \eta)$ as the number of inter-cluster edges between them. As more inter-cluster edges indicate that more complex patterns respectively in the given two clusters have common subsets of event types, preferentially examining the cluster pairs with the higher similarity can thus help identify more common sub-patterns by DCE within constrained construction time.

For a cluster (internal vertex), the residual instances in it are the instances of the differential parts of its child vertices.

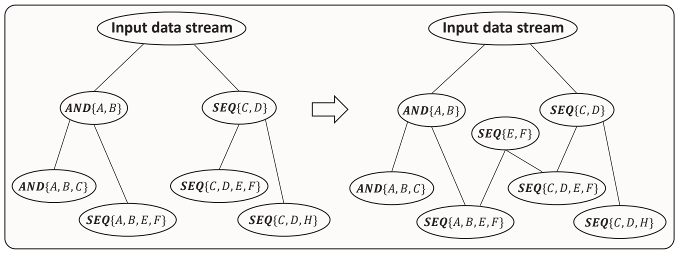


Fig. 8: An example of the differential compensation examination. Here, ξ and η refer to the vertices with $\mathbf{AND}\{A, B\}$ and $\mathbf{SEQ}\{C, D\}$; $\mathbf{SEQ}\{E, F\}$ is the common sub-pattern of the differential parts of their child vertices with $\mathbf{SEQ}\{A, B, E, F\}$ and $\mathbf{SEQ}\{C, D, E, F\}$.

By this, we resolve the examination of a pair clusters (*e.g.*, (ξ, η)) into examining each pair of the child vertices of them. Here, only the child vertex pairs with inter-cluster edges deserves to be examined. In the examination of each pair of child vertices (lines 7-11 in Algorithm 2), if there exists the common sub-pattern between their differential parts, DCE creates a new parent vertex for them marked by the identified common sub-pattern (as shown in Fig. 8). Essentially, if given enough plan construction time, DCE is implemented in an exhaustive manner that examines all the cluster pairs with inter-cluster edges; even so, as it operates at the cluster level, it is still lightweight.

In summary, the DCE algorithm consists of three steps: (1) computing the similarities of the cluster pairs in CCL, which can also be executed in parallel; (2) examining the child vertices of the cluster pair with the highest similarity; (3) updating $\mathbf{AP}$ and CCL, and returning to step (2).

5.2 Complexity analysis of multi-level aggregation plan construction

The aggregation plan construction consists of the CASEM and DCE algorithms. The first step of CASEM is building the correlation graph of complex patterns, which needs to compute the similarities of each pair of complex patterns and brings a complexity of $O(|\mathcal{W}|^2)$. Then, CASEM computes the entropy reductions of each pair of vertices (with edges) in the current aggregation plan, and preferentially examines the pairs with higher entropy reductions. Let M denote the maximal number of the neighbors of any node in correlation graph, the number of vertex pairs being examined in CASEM is upper bounded by $O\left(M|\mathcal{W}|\right)$.

In each examination, CASEM identifies the common sub-pattern of the given vertex pair from their sub-patterns. As the sub-patterns of a complex pattern essentially consist of the subsets of its required primitive event types, as well as the requirements on logic, time, and attributes, each examination results in a maximum complexity of $O\left(2^L\right)$, where L denotes the maximum pattern length in $\mathcal{W}$. Obviously, each examination is expensive in terms of the complexity, which exponentially increases with the pattern length. By preferentially examining the closely linked complex pattern clusters, CASEM can concentrate limited computational resources on more "valuable" examinations and thus speed up the aggregation plan construction. The complexity of CASEM is upper bounded by $O\left(|\mathcal{W}|^2 + 2^L M|\mathcal{W}|\right)$.

After completing CASEM, PatternInsight then moves to execute DCE. DCE is implemented by examining the child vertex pairs of the non-leaf vertex (cluster) pairs in the aggregation plan returned by CASEM. As the number of clusters in a graph is smaller than the number of nodes, the complexity of DCE is upper bounded by $O\left(2^L M|\mathcal{W}|\right)$.

Note that we here do not consider the complexity of acquiring stream statistics, which is required by existing approaches. As introduced in Section 2, existing approaches mainly focus on the single-level instance aggregation. When optimizing the single-level aggregation plan, one important issue is to resolve the conflicts among sub-patterns; that is, a complex pattern can share common sub-patterns with different ones, and such common sub-patterns have overlaps. For example, $\mathbf{SEQ}\{A, B, C, D\}$ ($\mathcal{P}_1$) can share $\mathbf{SEQ}\{A, B, C\}$ with $\mathbf{SEQ}\{A, B, C, E\}$ ($\mathcal{P}_2$), and also can share $\mathbf{SEQ}\{B, C, D\}$ with $\mathbf{SEQ}\{B, C, F\}$ ($\mathcal{P}_3$).

In the above example, for single-level aggregation approaches, they have to make a choice among the following three plans: (1) sharing $\mathbf{SEQ}\{A, B, C\}$ between $\mathcal{P}_1$ and $\mathcal{P}_2$; (2) sharing $\mathbf{SEQ}\{B, C, D\}$ between $\mathcal{P}_2$ and $\mathcal{P}_3$; (3) sharing $\mathbf{SEQ}\{B, C\}$ among $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$. To this end, existing approaches evaluate the expected instance size reductions of such three plans based on stream statistics, and choose the optimal one. Such statistically directed manner results in a complexity of $O(2^L|\mathcal{W}|^2 S)$, where S is the complexity of acquiring stream statistics which relies on a long-time observation and consuming statistical analysis of data streams.

Fortunately, with the multi-level instance aggregation in PatternInsight, we can take the overlapping part as a higher-level vertex in the aggregation plan, thus not relying on the stream statistics. Therefore, by leveraging the latent hierarchical cluster structure among complex patterns, PatternInsight can (1) avoid wasting computing resources on examining the vertex pairs which have little aggregatable instances, and (2) eliminate the cost of acquiring stream statistics. Such two advantages of PatternInsight make aggregation plan construction fast and scalable.

Discussion. The potential scalability bottleneck of PatternInsight lies in the correlation graph construction (with a complexity of $O(|\mathcal{W}|^2)$). It essentially is implemented via a double loop; it traverses the complex patterns in the workload $\mathcal{W}$, and for each complex pattern, computes its similarities with other $(|\mathcal{W}|-1)$ ones; the number of similarity computations is $\frac{|\mathcal{W}|(|\mathcal{W}|-1)}{2}$. In practice, when facing an extremely large workload, the correlation graph construction can be implemented in a distributed manner— by dividing complex patterns into multiple groups (uniformly or nonuniformly) and executing the similarity computation of each group on different compute units. Given that there are N compute units and the group dividing is uniform, the complexity at an unit is $\frac{|\mathcal{W}|(|\mathcal{W}|-1)}{N \cdot 2}$.

5.3 Implementation of multi-level instance aggregation

The CASEM and DCE algorithms return a tree-like multi-level aggregation plan to the CEP system for implementing instance aggregation during real-time detection. For taking advantage of the multi-level aggregation plan, PatternInsight creates candidate instances in a top-to-down manner (as shown in Fig. 3). Concretely, with the input data streams, each type of primitive event is inserted into a specific sharable buffer, so that the detection process of each pattern can call the primitive events in the buffer to create candidate instances. Then, over the multi-level aggregation plan, from the third level, PatternInsight will not execute the candidate instance creation of a vertex until there emerge matched instances of its all parent vertices. By this, PatternInsight can avoid repeatedly creating and storing the instances of common sub-patterns at all levels. The detailed procedures for the top-to-down instance creation are in Appendix B in the supplemental material.

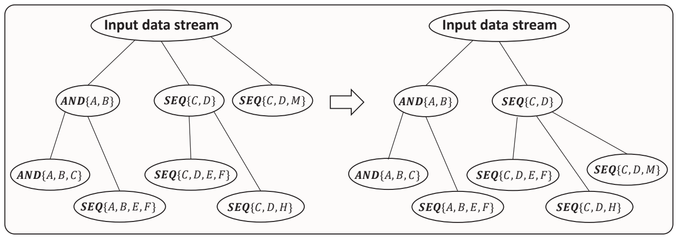


Fig. 9: An example of the in-stream adaptation. Here, the new complex pattern with $\mathbf{SEQ}\{C, D, M\}$ is merged into the cluster tagged by $\mathbf{SEQ}\{C, D\}$.

5.4 In-stream adaptation

During the real-time complex pattern detection with the aggregation plan produced by PatternInsight, when new complex patterns are defined, the aggregation plan then requires incremental updates. For a new complex pattern, we first add it as a new node into the correlation graph and link it with its similar complex patterns (nodes). At the same time, we add such a new complex pattern as a new leaf vertex to the current aggregation plan (as shown in the left part of Fig. 9). Based on this, we re-execute CASEM and DCE over the updated aggregation plan and correlation graph; here, we only need to locally examine the vertex pairs which contain the newly added complex patterns. After finishing the CASEM and DCE algorithms, the CEP system then adopts the updated aggregation plan.

6 Experiments

We evaluate the performances of PatternInsight based on the data streams generated in real-world mobile applications. The experiments aim at investigating the two key questions: (1) Can PatternInsight construct the aggregation plan more fast and scalably comparing with the state-of-the-art methodologies? (2) Can the aggregation plan produced by PatternInsight yield higher detection efficiency?

6.1 Experimental setup

Datasets and workloads. We conduct experiments on two mobile data streams which respectively record the time-ordered vehicle and sensor network data in a city.

(1) *Vehicle*. The Vehicle data stream is from CityPulse [29], [30], an open IoT dataset platform which collects data via crowdsensing. This data stream consists of 13M data items, which monitor the numbers of vehicles and traveling speeds at 449 intersections in a city. We define four

workloads respectively of 100, 500, 1000, and 2000 complex patterns about traffic situations, and define the primitive events based on abnormal vehicle numbers and/or traveling speeds. The attribute requirements further specify the ranges of vehicle numbers and speeds.

(2) *Sensor*. The Sensor data stream is also from CityPulse [29], [30], and monitors a set of pollution indices at 449 locations in a city with distributed sensors, including PM, CO, NO_2, SO_2, *etc.* There are 7.8M data items in this data stream. We define four workloads, each of 100, 500, 800, and 1000 complex patterns, where the primitive events are defined as the abnormal values of a specified index at a location; the attribute requirements further specify the ranges of index values.

Baselines. We compare the performance of PatterInsight with the following two representative state-of-the-art methodologies of instance aggregation (*i.e.,* combinatorial optimization-based and heuristic ones), which are mainly adopted by current complex pattern detection solutions.

(1) *SPASS*. The SPASS framework constructs the single-level aggregation plan in the statistically directed manner. SPASS first traverses sub-patterns of given complex patterns to identify the common sub-patterns that are shared by more than one complex pattern. For each common sub-pattern, SPASS estimates its total instance size when complex patterns are separately detected (*i.e.,* before aggregation), as well as the instance size of such a common sub-pattern if aggregating its instances among the complex patterns that share it (*i.e.,* after aggregation); both of the instance sizes before and after aggregation are estimated based on stream statistics. Then, SPASS evaluates the expected instance size reduction of each common sub-pattern with the ratio between its instance sizes before and after aggregation. Further, based on the expected instance reductions of common sub-patterns, SPASS seeks the aggregation plan with minimum instance size through combinatorial optimization. SPASS is designed by Ray *et al.* in [16], and is also adopted by the solutions in [6], [26], [31], [32], *etc.*

(2) *RW*. We mark the ideas initially proposed in [15] as RW, which speeds up the statistically directed construction manner via random walk. That is, RW implements aggregation plan construction with a random walk process over a graph, where each complex pattern links those sharing common sub-patterns with it. During each step of the random walk, RW moves into a pair of linked complex patterns and examines their sub-patterns; if identifying a sub-pattern that aggregating its instances can further reduce the overall instance size of given workload, RW updates the aggregation plan accordingly. The advantage of RW is that it can return the current-best solution at any time. Multiple recent work [7], [21], [33] also adopt the idea of RW.

Besides SPASS and RW, although plenty of work focuses on facilitating complex pattern detection, their contributions are mostly orthogonal to instance aggregation. Specifically, the parallel detection mechanisms are proposed in [6], [7], [9], [12], which distributedly create and store the instances of sub-patterns at different devices. [26] and [32] discuss the time when to start creating the instances of sub-patterns/ complex patterns. [13] and [38] design approximate yet efficient detection frameworks respectively based on load shedding and deep learning. The above mechanisms can be integrated with SPASS, RW, and PatternInsight if powered by multiple computing units and/or in specific applications (*e.g.,* allowing approximate detection). Thus, we here compare the performances of PatternInsight with the two representative baselines SPASS and RW on instance aggregation. We will review the places of SPASS, RW, and PatternInsight in the literature in more detail in Related Work (Section 8).

Evaluation metrics. The complex pattern detection solutions based on instance aggregation care about two aspects of performance: (1) the time costs for aggregation plan construction, and (2) the efficiency during real-time detection [15], [16]. For the former, we adopt the metric *construction time* that records the time cost on constructing the aggregation plan. Further, we adopt two metrics for evaluating the detection efficiency; one is the *peak memory cost*, which reflects the size of candidate instances; the other is the average *processing time* [15] of per unit of data items, which reflects the throughput of CEP system. As each item arrives, the processing time is spent on creating new instances and checking if they matching any vertex in the aggregation plan. Through optimizing these metrics, the performances of CEP systems such as on CPU utilization, communication latency and power consumption can also be improved.

Environments. We implement all algorithms in Python, and conduct experiments on a computer running Ubuntu 20.04 LTS with 48 cores 2.10 GHz (Intel Xeon Silver 4116) and 128 GB memory, which are the typical configurations of an edge server for high-volume streaming data processing.

Code and data. The code and data involved in this paper are available at https://github.com/ryy980622/PatternInsight.

6.2 Performance evaluation of aggregation plan construction

Table 1 presents the aggregation plan construction time of different approaches. Throughout this section, PatternInsight (CASEM-2) denotes the cases in which we use CASEM to construct an aggregation plan with two levels of instance aggregation; PatternInsight (CASEM) refers to the implementation of CASEM until its stopping condition is met; PatternInsight (CASEM+DCE) represents the use of CASEM to construct the aggregation plan together with DCE.

In experiments, we execute SPASS and the algorithms in PatternInsight 10 times, finding that the construction processes of them are the same each time. This is because SPASS leverages combinatorial optimization that constructs the aggregation plan greedily with the sub-patterns having the largest instance size reductions; also, PatternInsight preferentially examines the vertex pair with the maximum entropy reduction (in DCE is the highest similarity), resulting in a deterministic construction process.

For RW, we also execute it 10 times, where the random walk yields different construction processes. However, we execute RW in an exhaustive manner—conducting the random walk-based sub-pattern examination among complex patterns until there is no common sub-pattern that can further reduce the instance size; by this, we can obtain the best aggregation plan produced by RW. Since all the edges in the graph in RW are visited, the final construction time and the produced aggregation plan are the same each time as

	Construction time on Vehicle (s)				Construction time on Sensor (s)			
Algorithms	$\lvert\mathcal{W}\rvert = 100$	500	1000	2000	$\lvert\mathcal{W}\rvert = 100$	500	800	1000
SPASS	113	2452	9524	–	168	265	331	469
RW	100	2963	9052	–	114	275	498	750
PatternInsight (CASEM-2)	1.3	5	63	626	1.1	4	21	38
PatternInsight (CASEM)	1.4	22	248	3350	1.3	14	68	105
PatternInsight (CASEM+DCE)	1.6	25	255	3370	1.4	17	82	148

Table 1: Evaluation plan construction time (s) under different workload sizes.

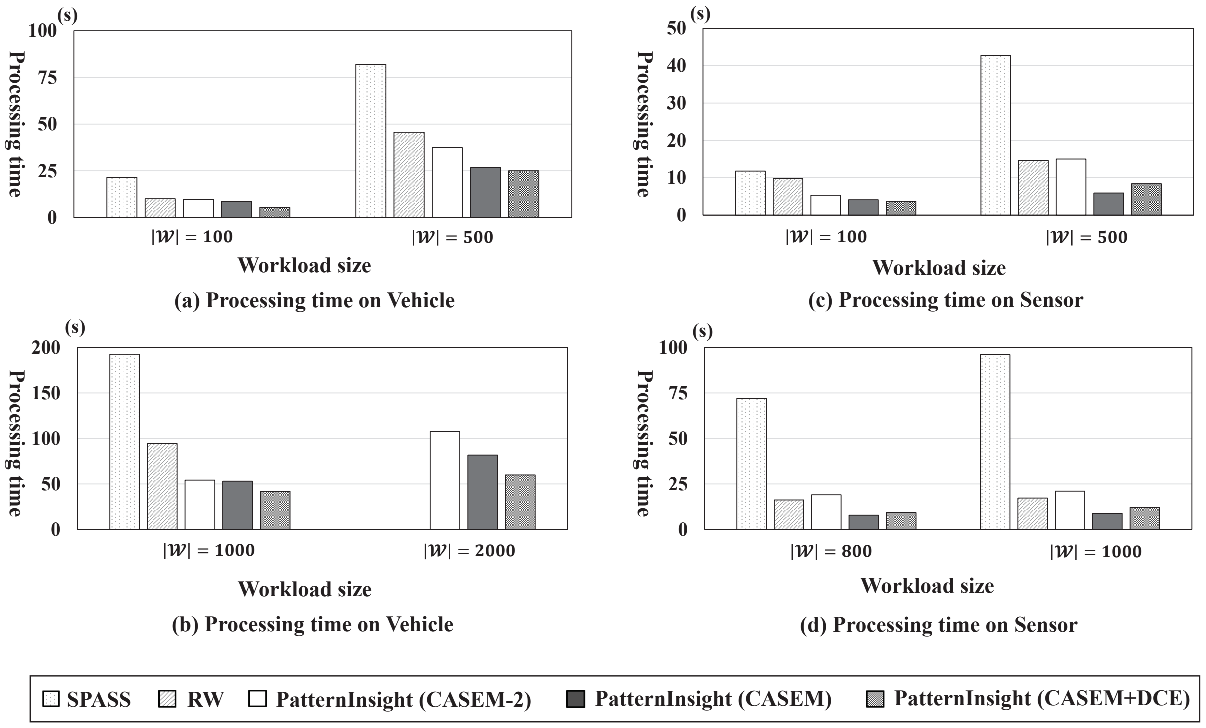


Fig. 10: The average processing time of per unit of data items under different workload sizes.

well. We then make comparisons between the aggregation plan produced by PatternInsight and the best one of RW regarding the detection efficiency performance.

As presented in Table 1, given the same workload, the PatternInisght approach can construct the multi-level aggregation plan much more efficiently, comparing with the two baselines which merely construct the single-level aggregation plan. The PatternInisght approach cuts the construction time by 90% on average, and up to 98% on Vehicle. The fast construction of PatternInisght approach benefits from the guidance of structural information among complex patterns. Since PatternInisght preferentially examines the closely linked complex patterns over correlation graph for identifying aggregatable instances, it can avoid wasting time costs on the sub-patterns with few instance reductions. Besides, as discussed in Section 5.2, PatternInsight can accommodate the sub-patterns with overlaps by multi-level instance aggregation, so that PatternInsight does not need to obtain the stream statistics; this further speeds up the aggregation plan construction. As presented in Section 5.2, the complexity of the aggregation plan construction in the two baselines is $O(2^L|\mathcal{W}|^2S)$, while for PatternInsight, it is upper bounded by $O\left(|\mathcal{W}|^2 + 2^L M|\mathcal{W}|\right)$.

The PatternInsight approach has a larger advantage on the construction time over Vehicle data stream; also, the construction time over Vehicle data stream in Table 1 is generally longer than Sensor. The reason is that, on Vehicle, we include the abnormal vehicle numbers and traveling speeds at a same intersection into one primitive event type, since the anomalies of such two indices usually emerge together (*e.g.*, too many vehicles jamming at an intersection will lower down their traveling speeds); whereas, on Sensor, we define the anomalies of different environmental indices at a same location as different primitive event types. As a result, the primitive event types on Vehicle are much less than on Sensor; given the same number of complex patterns defined on Vehicle and Sensor, there are more common sub-patterns among those on Vehicle. Thus, Vehicle has a larger search space for identifying aggregatable instances. Owing to the efficient sub-pattern examination in PatternInsight, it saves much more construction time over the Vehicle data stream comparing with the baselines.

In Table 1, we can also find that the construction time of CASEM is significantly longer than CASEM-2; this indicates that the solution (search) space is enlarged by times when considering multi-level instance aggregation. Then, from

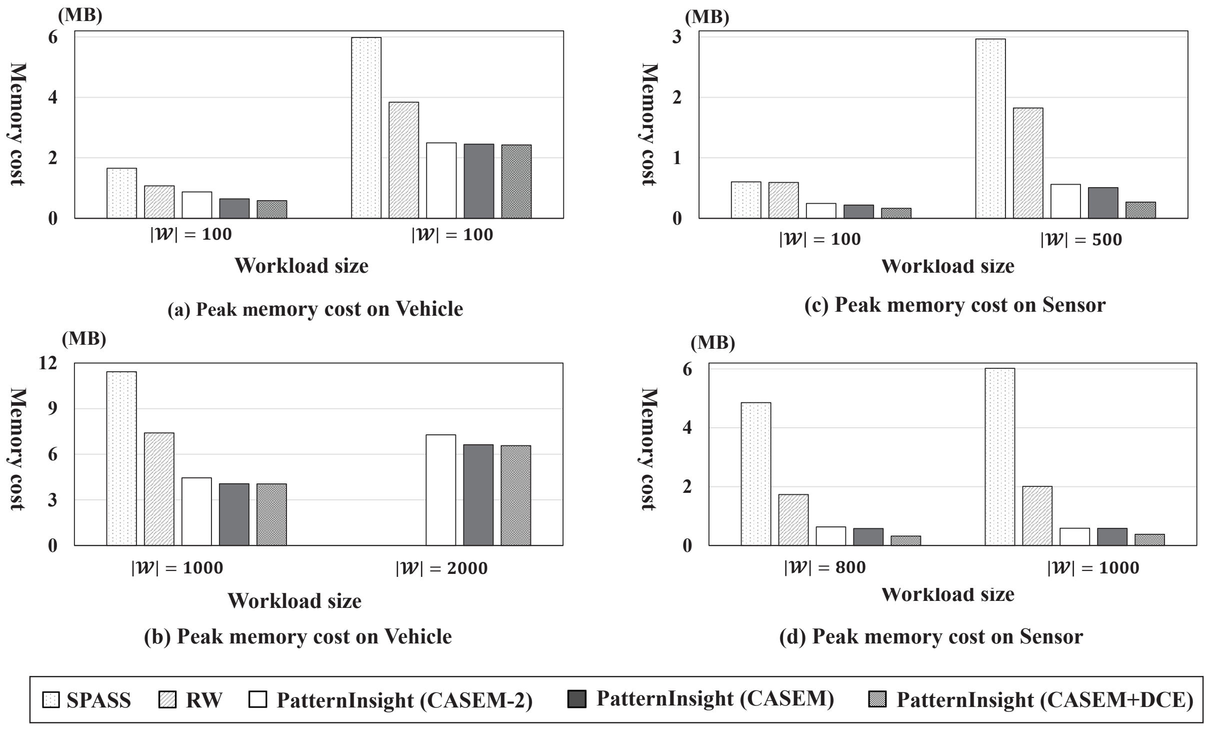


Fig. 11: The peak memory cost during detection under different workload sizes.

CASEM to CASEM+DCE, there is merely a slight rise in construction time, justifying that the DCE algorithm based on cluster-level search is lightweight. The comparisons between CASEM, CASEM-2, and CASEM+DCE serve as the ablation study of our designs—CASEM, CASEM for multi-level aggregation plan construction, and DCE. Moreover, the increase rate of the construction time from CASEM to CASEM+DCE over Sensor is higher than that over Vehicle. The underlying reason is that, as the primitive event types on Sensor are much more than Vehicle, the aggregation plan produced by CASEM over Sensor consists of clusters with a comparatively large number but small sizes; this makes the cluster-level search over Sensor cost more time.

Further, when implementing RW, we conduct the random walk-based sub-pattern examination in an exhaustive manner, inducing that the construction time of RW is longer than SPASS (in Table 1). Due to the prohibitive construction time of RW and SPASS, we do not implement them over the workload with 2000 complex patterns.

6.3 Performance evaluation of complex pattern detection

We then evaluate the efficiency of the online detection respectively over the aggregation plans produced by SPASS, RW, and PatternInsight. Figs. 10 and 11 respectively present the average processing time per unit of data items and the peak memory costs of different approaches during detection. We define the data items generated within one day as one unit in Vehicle and Sensor. When a primitive event arrives, the CEP system needs to create new candidate instances which contain such an event, and then traverses currently stored instances to check if there exist matched instances of any vertex in aggregation plan. After completing the above processing procedures of current event, we then insert next one into the CEP system. Through optimizing the processing time, we can speed up the instance detection and thus minimize the communication delay of CEP systems. Meanwhile, through optimizing the memory cost, we can also relieve the CPU utilization and power costs, especially when CEP systems are centrally deployed at edges, clouds or data centers.

It can be seen from Figs. 10 and 11 that, PatternInsight has much less processing time and memory cost comparing with the two baselines (mostly less than 60%), benefiting from the multi-level instance aggregation. Even for CASEM-2 which relies on the two-level instance aggregation, its detection efficiency is already much higher than the two baselines with single-level aggregation. Particularly, PatternInsight (CASEM+DCE) has the least memory cost among all the five approaches, since it not only leverages the multi-level instance aggregation but also further reduces the instance size by DCE. However, the processing time of PatternInsight (CASEM+DCE) is sometimes longer than PatternInsight (CASEM) as shown in Fig. 10. This is because there are more internal vertices in the aggregation plan constructed by PatternInsight (CASEM+DCE), and each internal vertex costs some processing time for checking if there emerging its matched instances. The similar phenomenon also exists in Fig. 10 (d), where the processing time of PatternInsight (CASEM-2) is a little longer than RW. For the two baselines, the processing time and memory cost of RW are less than SPASS, since we conduct RW in an exhaustive manner for obtaining its best aggregation plan. In summary, the results in Figs. 10 and 11 demonstrate that PatternInsight can significantly improve the efficiency of complex pattern detection over data streams.

7 Discussion

In this paper, we consider the primitive events that are defined based on geographical locations where the corresponding data items are generated, and implement instance aggregation with the geographical proximities among complex patterns. For other types of data streams (*e.g.*, security alerts and stock prices), our designs are also applicable; in such streams, the primitive events can be defined based on device IPs and stock IDs; as long as there exist multiple complex patterns that care about same primitive events, there are chances for instance aggregation. Further, for the workloads where event similarity is sparse (*i.e.*, do not have many common sub-patterns), one solution for utilizing instance aggregation to improve detection efficiency can be aggregating primitive events (*e.g.*, defining primitive events within an area or an IP range, instead of targeting on specific locations or IP addresses).

Moreover, the mobile data streams in practice are possibly noisy and subject to failures or missing data. To cope with such robustness issues, one candidate solution is introducing the "redundant" patterns. For example, the $\mathbf{NOT}$ (*i.e.*, !) operator can be used to characterize possible missing data. For the pattern with $\mathbf{SEQ}\{A, B, C, D\}$, we can use the redundant pattern $\mathbf{SEQ}\{A, B, !C, D\}$ to characterize the possible missing data at location C. Then, for the possible data noise (*e.g.*, sensor data acquisition error), we can learn noise distributions and define redundant patterns with specific attribute requirements. Such redundant patterns can be predefined into the workload, or added into the aggregation plan later on by in-stream adaptation. The method for in-stream adaptation is given in Section 5.4. Besides, the deep learning-based *approximate* detection mechanism in [38] can also be adopted over noisy streams.

8 Related work

In this section, we review prior studies on complex pattern detection over data streams, and introduce related work on the applications of structural information theory.

Complex pattern detection over data streams. For detecting complex patterns, the early CEP systems adopt the NFA mechanism that independently creates candidate instances of each complex pattern [15], [17], [18], [34]. Considering the efficiency issue caused by huge candidate instance sizes, the instance aggregation approach is widely adopted to aggregate common partially matched instances among multiple complex patterns [35], [36]. The early instance aggregation solutions are limited to aggregating the instances of common prefixes of complex patterns [15].

Then, Mei *et al.* [20] decompose each complex pattern into multiple sub-patterns, in order to reduce the instance size through prefiltering invalid instances of sub-patterns before combining them to form candidate instances of complex patterns. This is the prototype of tree-like aggregation plan. As aggregating the instances of sub-patterns represented by tree vertices enables flexible aggregation (not limited to prefixes), the tree-like plans become the mainstream in CEP systems in the past decade [16], [22], [26], [37].

For seeking the optimal aggregation plan with minimum instance size, these studies adopt a statistically directed construction manner, which involves a traverse of sub-patterns for identifying aggregatable instances. Taking such aggregatable instances as input, [16] formally formulates aggregation plan construction into an NP-hard combinatorial optimization problem, and designs an approximate solution with performance guarantee. Then, Maschi *et al.* [31] apply the solution in [16] into a flight search system where complex patterns are defined with business rules. The parallel detection mechanisms are proposed in [6], [9], [12], which distributedly deploy the branches of the tree-like plan in [16] at multiple devices. Besides, [26] and [32] combine the solution in [16] with the reordering method that will not create instances until there emerges the last primitive event in $\mathbf{SEQ}$ operators and the event with the least frequency in $\mathbf{AND}$, thus reducing kept candidate instances.

Although the approximate solution in [16] can construct the aggregation plan in polynomial time, the preliminary step of identifying aggregatable instances remains the scalability bottleneck. Recently, [15] speeds up the statistically directed manner through a random walk-based examination over a graph where each complex pattern links those sharing common sub-patterns, which makes a trade-off between the plan quality and construction time. Then, [21] designs pruning methods over such a graph to cope with the search space enlarged by the complex patterns with Kleene closure, which can be characterized by multiple (nested) $\mathbf{OR}$ operators and attribute requirements. With the aggregation plan produced by [15], Akili *et al.* [7] implement online detection in a networked manner that detects sub-patterns at multiple edge devices and forwards instances to "sink" nodes. Also, [33] discusses how to adapt streaming log data generated by mobile applications to the aggregation plan in [15].

Further, [22] presents an adaptive aggregation plan optimization mechanism for dealing with fluctuating stream statistics. The approximate detection methods are proposed in [13] and [38] to seek efficiency improvements, which can also be used to cope with the noises in mobile data streams. The above distributed/ reordering/ adaptive/ approximate mechanisms can be integrated with SPASS, RW, and PatternInsight. Then, although [23] also considers the multi-level instance aggregation, it heavily depends on the conceptual hierarchies among events (*e.g.*, the events in UTD, Dallas, and Texas); whereas, we define sub-patterns based on the primitive event type subsets and compatible requirements, with a much wider applicability.

Applications of structural information theory. The structural information theory aims at minimizing the expected code length for encoding a random walk process over a graph (*i.e.*, structural entropy), through partitioning closely correlated nodes into hierarchical clusters [25]. Based on this, [39] links the gene expression profiles through minimizing the structural entropy among them, and then uncovers the types and sub-types of tumor genes based on the corresponding partitioning tree. With the similar idea, [28] decodes the domains of chromosomes from Hi-C data. Liu *et al.* [40] define a structural entropy-based metric for a task of hiding the community structure in social network under privacy concerns. In the CASEM algorithm designed in this paper, we take the closeness among nodes as the guidance to preferential examination; whether or not merge or combine them into clusters further depends on the nodes' contents represented by complex patterns.

9 Conclusion

This paper presents the PatternInsight approach for realizing the fast and scalable multi-level instance aggregation to make it well support the online complex pattern detection. PatternInsight leverages the latent hierarchical cluster structure as the guide information to speed up the multi-level instance aggregation plan construction. This is implemented by the content-aware structural entropy minimization algorithm that uncovers and preferentially examines the intra-cluster complex patterns for identifying aggregatable instances, together with the differential compensation examination algorithm for inter-cluster examination. The novel top-to-down instance creation mechanism is designed to fully leverage the multi-level instance aggregation plan during detection. The evaluation against state-of-the-art approaches demonstrates significant performance advantages of PatternInsight, which cuts the aggregation plan construction time by 90% on average (up to 98%) and saves the time and memory costs during detection by 40%; the performance advantage increases with the pattern length and number. The high scalability in construction, the low memory cost and processing time during detection pose PatternInsight as a practical solution for complex pattern detection in large-scale mobile applications.

References

[1] Z. Dai, C. H. Liu, R. Han, G. Wang, K. Leung, and J. Tang. Delay-sensitive energy-efficient uav crowdsensing by deep reinforcement learning. *IEEE Transactions on Mobile Computing*, 2021.

[2] Y. Zhang, Z. Liu, R. Wang, T. Yang, J. Li, R. Miao, P. Liu, R. Zhang, and J. Jiang. Cocosketch: High-performance sketch-based measurement over arbitrary partial key query. In *Proc. ACM SIGCOMM*, pages 207–222, 2021.

[3] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani. Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):2923–2960, 2018.

[4] L. Tang, Q. Huang, and P. PC Lee. Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams. In *Proc. IEEE INFOCOM*, pages 2026–2034. IEEE, 2019.

[5] Y. Li, W. Liang, W. Xu, Z. Xu, X. Jia, Y. Xu, and H. Kan. Data collection maximization in iot-sensor networks via an energy-constrained uav. *IEEE Transactions on Mobile Computing*, 22(1):159–174, 2021.

[6] S. Akili, S. Purtzel, and M. Weidlich. Decopa: Query decomposition for parallel complex event processing. *Proc. ACM SIGMOD*, 2(3):1–26, 2024.

[7] S. Akili, S. Purtzel, and M. Weidlich. Inev: in-network evaluation for event stream processing. *Proc. ACM SIGMOD*, 1(1):1–26, 2023.

[8] X. Liu, J. Yin, J. Liu, S. Zhang, and B. Xiao. Time efficient tag searching in large-scale rfid systems: A compact exclusive validation method. *IEEE Transactions on Mobile Computing*, 21(4):1476–1491, 2020.

[9] S. Akili and M. Weidlich. Muse graphs for flexible distribution of event stream processing in networks. In *Proc. ACM SIGMOD*, pages 10–22, 2021.

[10] E. N. Budisusila, M. Khosyi'in, S. A. D. Prasetyowati, B. Y. Suprapto, and Z. Nawawi. Ultrasonic multi-sensor detection patterns on autonomous vehicles using data stream method. In *IEEE 8th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pages 144–150. IEEE, 2021.

[11] ScienceABC. What are phantom traffic jams? https://www.scienceabc.com/eyeopeners/what-are-phantom-traffic-jams.html, 2022.

[12] M. Yankovitch, I. Kolchinsky, and A. Schuster. Hypersonic: A hybrid parallelization approach for scalable complex event processing. In *Proc. ACM SIGMOD*, pages 1093–1107, 2022.

[13] Koral Chapnik, Ilya Kolchinsky, and Assaf Schuster. Darling: data-aware load shedding in complex event processing systems. *Proc. VLDB Endowment*, 15(3):541–554, 2021.

[14] R. Izadpanah, B. A Allan, D. Dechev, and J. Brandt. Production application performance data streaming for system monitoring. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 4(2):1–25, 2019.

[15] I. Kolchinsky and A. Schuster. Real-time multi-pattern detection over event streams. In *Proc. ACM SIGMOD*, pages 589–606, 2019.

[16] M. Ray, C. Lei, and E. A. Rundensteiner. Scalable pattern sharing on event streams. In *Proc. ACM SIGMOD*, pages 495–510, 2016.

[17] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *Proc. ACM SIGMOD*, pages 407–418, 2006.

[18] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *Proc. ACM SIGMOD*, pages 147–160, 2008.

[19] B. Zhao, N. Q. V. Hung, and M. Weidlich. Load shedding for complex event processing: Input-based and state-based techniques. In *Proc. ICDE*, pages 1093–1104. IEEE, 2020.

[20] Y. Mei and S. Madden. Zstream: a cost-based query processor for adaptively detecting composite events. In *Proc. ACM SIGMOD*, pages 193–206, 2009.

[21] L. Ma, C. Lei, O. Poppe, and E. A Rundensteiner. Gloria: Graph-based sharing optimizer for event trend aggregation. In *Proc. ACM SIGMOD*, pages 1122–1135, 2022.

[22] O. Poppe, C. Lei, L. Ma, A. Rozet, and E. A Rundensteiner. To share, or not to share online event trend aggregation over bursty event streams. In *Proc. ACM SIGMOD*, pages 1452–1464, 2021.

[23] M. Liu, E. Rundensteiner, K. Greenfield, C. Gupta, S. Wang, I. Ari, and A. Mehta. E-cube: multi-dimensional event sequence analysis using hierarchical pattern query sharing. In *Proc. ACM SIGMOD*, pages 889–900, 2011.

[24] Y. Song, R. C. W. Wong, and X. Zhao. Speech-to-sql: toward speech-driven sql query generation from natural language question. *The VLDB Journal*, pages 1–23, 2024.

[25] A. Li and Y. Pan. Structural information and dynamical complexity of networks. *IEEE Transactions on Information Theory*, 62(6):3290–3339, 2016.

[26] I. Kolchinsky and A. Schuster. Join query optimization techniques for complex event processing applications. *Proc. VLDB Endowment*, 11(11):1332–1345, 2018.

[27] M. Li, H. Dai, X. Wang, R. Xia, A. X Liu, and G. Chen. Thresholded monitoring in distributed data streams. *IEEE/ACM Transactions on Networking*, 28(3):1033–1046, 2020.

[28] A. Li, X. Yin, B. Xu, D. Wang, J. Han, Y. Wei, Y. Deng, Y. Xiong, and Z. Zhang. Decoding topologically associating domains with ultra-low resolution hi-c data by graph structural entropy. *Nature communications*, 9(1):1–12, 2018.

[29] CityPulse Dataset Collection. http://iot.ee.surrey.ac.uk:8080/datasets.html.

[30] M. Ali A. Mileo M. Hauswirth F. Ganz S. Ganea B. Kjærgaard D. Kuemper S. Nechifor D. Puiu A. Sheth V. Tsiatsis L. Vestergaard R. Tönjes, P. Barnaghi. Real time iot stream processing and large-scale data analytics for smart city applications, 2014.

[31] F. Maschi, M. Owaida, G. Alonso, M. Casalino, and A. Hock-Koon. Making search engines faster by lowering the cost of querying business rules through fpgas. In *Proc. ACM SIGMOD*, pages 2255–2270, 2020.

[32] X. Hu, S. Sintos, J. Gao, P. K. Agarwal, and J. Yang. Computing complex temporal join queries efficiently. In *Proc. ACM SIGMOD*, pages 2076–2090, 2022.

[33] I. Mavroudopoulos and A. Gounaris. Siesta: A scalable infrastructure of sequential pattern analysis. *IEEE Trans. on Big Data*, 9(3):975–990, 2022.

[34] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, 44(3):1–62, 2012.

[35] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm. A catalog of stream processing optimizations. *ACM Computing Surveys*, 46(4):1–34, 2014.

[36] M. Akdere, U. Çetintemel, and N. Tatbul. Plan-based complex event detection across distributed sources. *Proc. VLDB Endowment*, 1(1):66–77, 2008.

[37] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig. Elastic sketch: Adaptive and fast network-

wide measurements. In *Proc. ACM SIGCOMM*, pages 561–575, 2018.

[38] A. Amir, I. Kolchinsky, and A. Schuster. Dlacep: A deep-learning based framework for approximate complex event processing. In *Proc. ACM SIGMOD*, pages 340–354, 2022.

[39] A. Li, X. Yin, and Y. Pan. Three-dimensional gene map of cancer cell types: Structural entropy minimisation principle for defining tumour subtypes. *Scientific reports*, 6(1):1–26, 2016.

[40] Y. Liu, J. Liu, Z. Zhang, L. Zhu, and A. Li. Rem: From structural entropy to community structure deception. In *Proc. NeurIPS*, pages 12938–12948, 2019.

Xudong Wu (Member, IEEE) received the Ph.D. degree from Shanghai Jiao Tong University, in 2021, and the B. E. degree from Nanjing Institute of Technology, in 2015. From 2021 to 2023, he was a Post-Doctoral Fellow at the School of Software, Tsinghua University. He is currently an Associate Research Fellow at the School of Computer Science and Technology, East China Normal University. His research of interests are in the area of data-driven network analysis and optimization.

Yuyang Ren received the B.E. degree from Tianjin University, Tianjin, China, in 2020. He is now pursuing the Ph.D. degree at Shanghai Jiao Tong University, Shanghai, China. His research interests include supervised and self-supervised graph representation learning.

Zhenhua Li (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees from Nanjing University, in 2005 and 2008 respectively, and the Ph.D. degree from Peking University in 2013, all in computer science and technology. He is a Tenured Associate Professor with the School of Software, Tsinghua University. His research areas cover network measurement, mobile networking/virtualization, and cloud computing/gaming.

Fei Xu (Member, IEEE) received the BS, ME, and PhD degrees from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2007, 2009, and 2014, respectively. He is currently a full professor and the deputy dean of the School of Computer Science and Technology at East China Normal University, Shanghai, China. His research interests include cloud computing and datacenter, and distributed machine learning systems.

Yunhao Liu (Fellow, IEEE) received the B.Sc. degree from the Automation Department at Tsinghua University, an M.Sc. degree and a Ph.D. degree in computer science and engineering from Michigan State University. He is a Full Professor and the Dean of Global Innovation Exchange (GIX), Tsinghua University. His research interests include sensor network, IoT, RFID, distributed systems, and cloud computing.

Guihai Chen (Fellow, IEEE) received the B.S. degree from Nanjing University, the M.E. degree from Southeast University, and the Ph.D. degree from The University of Hong Kong. He is currently a Distinguished Professor with the Department of Computer Science, Shanghai Jiao Tong University. He has published over 200 papers in peer-reviewed journals and refereed conference proceedings in the areas of wireless sensor networks, high-performance computer architecture, peer-to-peer computing, and performance evaluation. He has served on technical program committees of numerous international conferences.