Event Regularity and Irregularity in a Time Unit

Lijian Wan
University of Massachusetts, Lowell
lwan@cs.uml.edu

Tingjian Ge
University of Massachusetts, Lowell
ge@cs.uml.edu

***Abstract*—In this paper, we study the problem of learning a regular model from a number of sequences, each of which contains events in a time unit. Assuming some regularity in such sequences, we determine what events should be deemed irregular in their contexts. We perform an in-depth analysis of the model we build, and propose two optimization techniques, one of which is also of independent interest in solving a new problem named the Group Counting problem. Our comprehensive experiments on real and hybrid datasets show that the model we build is very effective in characterizing regularities and identifying irregular events. One of our optimizations improves model building speed by more than an order of magnitude, and the other significantly saves space consumption.**

I. Introduction

We live in "interesting times". For computing, this is true in that we are inundated with data, and computing power is ever increasing. Much of this data has to do with ubiquitous computing — data given by sensors that detect various attributes of the state of the world, e.g., GPS (for locations), temperature and light sensors, accelerometers and gyroscopes (for movements and orientations), among many others that even come with smartphones today. Continuous sensing creates relational sequence data where one of the attributes is *time*, along with any number of other attributes. Each tuple in the sequence describes the state at an instant of time, which often implies the event, activity, or action that is occurring. Certain events and activities often have some patterns within a *time unit* (typically a day or the duration of a process), and the same pattern is repeated periodically over many time units. It is useful to find out the model of such regularity, and to identify irregular or unusual events when they happen. Let us look at some examples.

Example 1: On March 24, 2015, the copilot Lubitz crashed Germanwings Flight 4U9525 in the French Alps, killing all 150 people on board. Data shows that in the flight into Barcelona prior to the doomed flight, Lubitz practiced a controlled descent, where he toyed with the plane's settings, programming it for sharp descent multiple times [1]. The "selected altitude" of the flight changed repeatedly, including several times being set as low as 100 feet above the ground. Lubitz also put the engines on idle, which gives the plane the ability to quickly descend. It is highly unusual for a pilot to do these. The airline company technically could have known about Lubitz' apparent rehearsal on this previous flight, but *only if it had looked at the flight data in the short period while the plane was unloading and loading passengers in Barcelona*. There are various sensors in a modern airplane to record operations and settings of the pilot. Suppose a "regular" model for a single flight (which is the *time unit* mentioned above) had been built, irregular and unusual events during this flight (i.e., going up and down with the selected altitude) would have been automatically detected, and the tragedy in the subsequent flight could have been prevented.

Example 2: Elderly people (or persons with autism) who live alone often lead a dangerous life. If they fall down on the floor, or pass out in the bathroom, no one may notice it for a long time. It would be useful to continuously sense the location, movement, and other signals from the person, and have a software program automatically monitor the situation. These people's normal daily activities are usually simple, upon which a regular model may be built (the time unit here is one *day*) and abnormal events may be detected as soon as they happen. Many lives could be saved from such continuous automatic monitoring.

Example 3: Consider a person who lives a normal life with established and habitual daily activities, such as going to office for work, having meetings with some group of people, dining in a certain set of restaurants, entertaining at a number of places, and so on. Variations are possible, say, between workdays and weekends. Automatic recognizing and logging "irregular" events are often of interest. For instance, if a meeting takes place with someone who I do not usually interact with or at a place I do not typically visit, it may be useful to have this logged. Or if anything significant happens, I may look back to see if any recent irregular events might have caused this outcome.

Event detection and activity recognition have been well-studied in the ubiquitous computing and artificial intelligence communities (e.g., [2]). In the above motivating applications, each tuple at a time instant may have an event/activity label indicating the event or action that is happening. Each time unit — e.g., a flight in the pilot example or a day in the latter two examples — has a sequence of tuples, and we may have a number of such sequences, from which a regular model is to be learned. Using this regular model, irregular events are further identified. Outlier/anomaly detection has been studied for sequence data (cf. a survey [3]), temporal data (cf. a survey [4]), and in more general contexts (cf. a survey [5]). However, previous work is designed for different problems and is very different from ours. We discuss it in detail in Sec. VIII (related work) and Sec. VII (experiments).

Our Contributions. We begin with formally stating the problem, and proposing to cluster events into an event hierarchy tree. This abstracts events into higher level commonalities, which we call generalized events, to build into the regular model. In this way, even though specific events may differ, variations can be permitted, and the higher level common characteristics of the events are recognized in the model. Then we first study how to build a regular model.

978-1-5090-2020-1/16/$31.00 © 2016 IEEE

Our algorithm performs message passing progressively and builds the regular model at the same time. This framework is easily parallelizable. The resulting model characterizes events and their contexts. Next we devise a method that returns the *regularity score* of each event in a sequence, based on the model. Our experiments show that our model is very effective in identifying irregular events.

We further perform an in-depth analysis of property of the model graph. We adaptively estimate the probability of creating a new edge into the model graph during the model building process, based on a novel *two function* recursive approach, also incorporating runtime information obtained from actual data. This results in an optimization that we can stop the model building algorithm early when the new edge probability is very small. Our experiments show that this optimization is very effective. The performance improvement is dramatic, often over an order of magnitude. Moreover, there is virtually no loss in accuracy. The precision is 1, and the recall is 0.99 or higher for edges in the model graph.

Another optimization is to reduce memory consumption. We first abstract out a general problem called the *Group Counting Problem*, which is of independent interest. It is a generalization of the commonly known *set membership problem* in computer science. We devise a low memory footprint approximate solution named *blended bitmap set* (BBS). Our experiments show that with BBS our regular model building uses significantly less memory (e.g., one eighth) without losing much accuracy and model power. Finally, we carry out a comprehensive experimental study to examine our regular model properties, effectiveness in finding irregular events, and the efficiency of our algorithms. Our contributions can be summarized as follows:

- We identify and formulate an interesting problem with practical significance (Sec. II).
- We propose an algorithmic framework based on message passing and event hierarchies to build a regular model, which can be easily parallelized (Sec. III).
- We propose a method that returns the regularity of all events in a sequence, which is shown to be very effective in experiments (Sec. IV).
- We perform two optimizations: one is to estimate a statistic dynamically at runtime and can stop the execution early, while the other is a new compact approximate data structure to significantly save memory consumption (Sec. V, VI).
- We perform a systematic experimental study to evaluate our approaches (Sec. VII).

II. Problem Statement and Preprocessing

A. Problem Statement

We are given a set of event sequences $S = \{s_1, s_2, \ldots, s_m\}$ as input, where each sequence $s_i (1 \leq i \leq m)$ consists of a list of events within the duration of a *time unit*. Thus, the time units are repeated, and each time unit is a time interval, such as a day or the duration of a flight. The whole set S spans m time units. The events in s_i are denoted as $s_i[1]$,$s_i[2], \ldots$ Note that s_i's may be of different lengths. Each event $s_i[j]$ is a tuple with a number of attributes including a timestamp attribute, which is monotonically increasing within each sequence s_i.

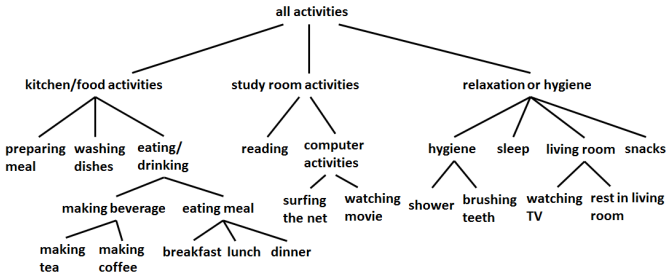

Fig. 1: Illustrating the event hierarchy tree created by Cobweb.

Given S, the *first aspect* of our problem is to obtain a model $\mathcal{M}$ that characterizes the regularity of the sequences in S. Most, if not all, sequences in S share this same model. We call it a *regular model*. The *second aspect* of our problem is to return a function $\mathcal{R}$, such that given the regular model $\mathcal{M}$ and any sequence s (regardless of whether $s \in S$), one can obtain the *regularity score* of any event in s, denoted as $R(s, j, \mathcal{M})$ for the jth event in s. Moreover, one can obtain the regularity score of the sequence s, denoted as $R(s, \mathcal{M})$. Such regularity scores are between 0 and 1, with 1 being the most regular and 0 the least. Note that, in the problem statement, we intentionally do not restrict the form of the regular model or the score function, in order to make the possible solutions flexible.

B. Preprocessing Events

For building a regular model, we propose to preprocess the events in the data into an event hierarchy tree $\mathcal{T}$. The idea is that a (new) generalized higher-level event incorporates multiple more specific events, hence allowing us to model variations of the same general event. For example, the events "drinking coffee" and "drinking tea" can be generalized to a higher level event "drinking beverage", even though "drinking beverage" is not an event in the original sequence set S (while "drinking coffee" and "drinking tea" are). Similarly, suppose a person sometimes has lunch at restaurant A, sometimes at restaurant B. Then the events "lunch at restaurant A" and "lunch at restaurant B" can be generalized into a higher level event "lunch at a restaurant", or even further, just "having lunch". All original events in S appear as *leaves* of the event tree $\mathcal{T}$, while internal nodes of $\mathcal{T}$ are higher-level events. Thus, if different sequences in S show variations of a general event, their commonality can still be modeled, knowing the event hierarchy $\mathcal{T}$. An event in that context that does not belong to the more general event can be an *irregular* event.

We merely propose the general strategy of clustering events into an event hierarchy $\mathcal{T}$. There can be multiple ways to implement this strategy. In fact, if nothing is done, all the original events in S can be deemed a "flat" tree, as leaves under a single root node. Hence, our subsequent algorithm framework is always applicable to any event data. Such clustering is based on the attributes in an event. We use a technique called *conceptual clustering* [6] in machine learning. Given a set of events (or object descriptions in general), it produces a classification scheme over the events based on their attributes. Specifically, we use the Cobweb algorithm provided as part of the commonly used Weka machine learning tools [7]. Fig.

1 shows a snippet of the event tree produced by Cobweb on one of our test datasets. After this conceptual clustering, each sequence s_i in S corresponds to a sequence of leaf nodes of $\mathcal{T}$. Each of these leaf nodes *e* belongs to more general event types that are *e*'s ancestors in $\mathcal{T}$.

III. Building a Regular Model

In this section, we propose an approach to solve the first aspect of the problem stated in Section II-A, namely to build a regular model. This takes as input a set of event sequences S and the event hierarchy tree $\mathcal{T}$ discussed in Section II-B. Intuitively, the general idea is that an event *e* is "regular" if it "fits in" its "context", in terms of what other events are before and after it. If this context is common for *e* to be in, then *e* is regular. However, one has to keep in mind that the relevant surrounding events in the context may not be *immediately* preceding or succeeding *e*, as some random events could occur in between.

Our algorithm is based on "message passing". Each event (in each sequence of S) initiates a message to its right neighbor event. These messages propagate one step at a time to the receiver's right neighbor. During this progressive propagation, each event pair $(message\_initiator, receiver)$ is added to a counter maintained for each event pair. When the counter for an event pair is above a threshold, this event pair is added as an edge in a *graph*, which we will eventually return as the regular model. The event pairs we keep track of include general event types as well, i.e., the internal nodes of $\mathcal{T}$. Thus, the model tolerates variations of events belonging to a general type, although an edge over a more general event type carries less weight when we compute the "regularity score" in Section IV.

Another point is that the regular model that we return is concise. During the message propagation, we do not add an edge from event e_1 to event e_2 to the model graph if there already exists a *path* from e_1 to e_2 in the model graph (which means each edge in this path takes fewer message passing steps). Thus, if there is a common event sequence $a \to b \to c$, then we only have two edges $a \to b$ and $b \to c$ in the model graph, but not $a \to c$. Besides efficiency, another reason for doing this is that, in considering an event *e*'s "context", we would prefer events closer to *e*, even though we do allow interleaving irrelevant events.

For a node *v* in T, we define the *level* of *v*, denoted as $l(v)$, to be the maximum number of edges to traverse from *v* to a leaf that is a descendant of *v* (i.e., in *v*'s subtree). Thus, all leaves have a level of 0. We show the algorithm in BuildRegularModel.

In line 1, we initialize a model graph, where the vertices are just the nodes of $\mathcal{T}$, and the edges are to be determined in this algorithm. In line 2, we start an "agent" for each distinct leaf event. These agents will be responsible for sending and receiving messages. In lines 3-4, we use the keyword "in parallel" to emphasize the fact that each loop can be executed independently in parallel. Indeed, our message passing algorithmic framework is highly parallelizable. For event $s[i]$ in sequence *s*, we denote its agent as $a[i]$. In line 5, $a[i+1]$ is the agent for the event's right neighbor.

Algorithm 1: BuildRegularModel $(S, \mathcal{T})$

Input: S: a set of m event sequences,
$\mathcal{T}$: an event hierarchy tree
Output: a regular model $\mathcal{M}$

```
1  G ←(V, E) where V is nodes of T and E is to be determined
2  set of agents A ← leaves of T
3  for each s ∈ S in parallel do
4      for each event s[i] ∈ s in parallel do
5          agent a[i] initiates a message and sends to a[i + 1]
6  R(u, v) ← 0, ∀u, v ∈ V, u ≠ v
7  R(v, v) ← 1, ∀v ∈ V
8  for each time step t do
9      for each agent a in parallel do
10         if a receives a message msg then
11             a_0 ← msg's initiator
12             for each v_1 ∈ ancestors (a_0), v_2 ∈ ancestors(a)
               do
13                 if R(v_1, v_2) = 1 then
14                     continue
15                 set the bit in bm(v_1, v_2) for a's sequence
16                 if |bm(v_1, v_2)| > τ · m then
17                     E ← E ⋃ (v_1, v_2)
18                     l ← max(l(v_1), l(v_2))
19                     for each x ∈ V do
20                         for each y ∈ V do
21                             if max(l(x), l(y)) ≥ l and
22                             R(x, v_1) = R(v_2, y) = 1 then
23                                 R(x, y) ← 1
24     for each s ∈ S in parallel do
25         for each event s[i] ∈ s in parallel do
26             agent a[i] propagates its message to a[i + 1]
27 return G as regular model M
```

In lines 6-7, the binary matrix $R(u, v)$ is to keep track of "reachability" from vertex *u* to vertex *v* in the model graph G so far. More precisely, $R(u, v) = 1$ if there exists a path from *u* to *v* where every vertex on this path has a level no greater than $max(l(u), l(v))$. This is exactly the condition under which we do not create a new edge from *u* to *v*. Note that if there is a vertex on this path with a level greater than those of both *u* and *v*, it is still beneficial to add a direct edge from *u* to *v*, since it does not go through a more general event (which is a stronger relationship). Lines 6-7 are to initialize R such that a vertex is only reachable to itself.

At each time step (line 8), the receiver agent receives the message from the sender agent. We call it a *time step* here, which simply means the propagation of one hop of the messages. Line 11 gets the original initiator of the message (the event/agent) who creates the message in the beginning (i.e., line 5). In line 12, we iterate through each ancestor of a_0 and each ancestor of *a* in the input event tree $\mathcal{T}$, where a_0 and *a* correspond to leaves of the tree. If there is a path from v_1 to v_2 (lines 13-14), then we do not bother to create a new edge. Otherwise, in line 15, we "increment the counter" for this event pair v_1, v_2. However, this is not a simple counter, as each sequence in S can only be counted once. That is, we keep track of, out of the *m* sequences in total, how many have

encountered the event pair (v_1, v_2) in the message passing so far. Therefore, we use a bitmap $bm(v_1, v_2)$ of m bits (one bit for each sequence) as the "counter" for this event pair. In line 16, $|bm(v_1, v_2)|$ denotes the number of 1's in this bitmap. If it is above a threshold (fraction τ), we add this event pair as an edge into E of the model graph.

Lines 18-23 are to maintain the reachability matrix R. Since we have just added an edge (v_1, v_2) into the graph, we want to find all vertex pairs (x, y) that are changed in R. Clearly, if there is already a path (x, v_1) and a path (v_2, y) indicated in R (line 22), there now exists a path (x, y) due to the addition of edge (v_1, v_2). There is an additional condition that x and y must not both be at levels lower than l in $\mathcal{T}$ (line 21); this is required by the definition of $R(x, y)$ being 1, as discussed earlier.

Lines 24-26 simply propagate each message to the right, preparing for the next time step (back to the loop in line 8). The loop in line 8 ends when there are no more messages to process, i.e., all messages have propagated through the right ends of the sequences. When that happens, the graph G is returned as the regular model $\mathcal{M}$ (line 27).

Example 4: Let us look at a simple example to have a more concrete idea of how the algorithm proceeds. Suppose S has three sequences s_1, s_2, and s_3 all being "abc". That is, there are three leaf events a, b, and c. For simplicity, let us assume the tree $\mathcal{T}$ is flat and we do not consider internal nodes for now. Let the threshold parameter $\tau = 2/3$ (line 16). We show how we obtain the final model, in which there are two edges (a, b) and (b, c). In line 5, each sequence will have two messages $a \rightarrow b$ and $b \rightarrow c$. In lines 6-7, R is a 3-by-3 matrix where only the diagonal entries are 1. In line 10, agent for event b receives a message from the agent for event a. The bitmap $bm(a, b)$ in line 15 has three bits, one for each sequence. All three bits of $bm(a, b)$ are set to 1 since all three sequences encounter this event pair message. In line 16, it is above the threshold 2; hence the edge (a, b) is added to the edge set E of the model graph (line 17). Lines 18-23 will set the entry (a, b) in R to be 1. In the same manner, the other message $b \rightarrow c$ in this round will create an edge (b, c) in the model graph, and set the reachability entry $R(b, c)$ to be 1. Since both $R(a, b)$ and $R(b, c)$ are 1, lines 19-23 will set $R(a, c)$ to be 1 as well. Now lines 24-26 propagate the message initiated by a and received by b to its right neighbor c, for all three sequences. Then in the next time step (loop at line 8), c receives this message originally initiated by a. Thus, we are considering the event pair (a, c). However, line 13 finds that $R(a, c) = 1$; therefore (a, c) will not be added as an edge in the model graph. At this point, all messages have propagated to the right end of the sequences, and the graph G containing two edges (a, b) and (b, c) is returned as the model.

We can see that the message passing framework in BUILDREGULARMODEL can be easily parallelized. Except for the serial time steps in line 8, all the execution steps within each time step are highly parallelizable. A few comments are in place. First, we have to understand the complexity that the "preceding" and "succeeding" relevant events in the context might not *immediately* precede or follow the event in question. There can be irrelevant or irregular events in between. This is one source of "noise". Second, we allow another type of variation that events may agree on a more general type in the hierarchy of $\mathcal{T}$, if not the most specific event type (i.e., leaf level of $\mathcal{T}$). However, different penalty weights may be inflicted depending on the application. For example, variations on the duration of an event may be significant for some applications, such as in the living-alone elder people example (Example 2), where a long stay in the bathroom is irregular and must be alerted. Edge weights in the model graph will be discussed in Section IV when we compute regularity scores.

IV. REGULARITY SCORE AND IRREGULAR EVENTS

Given a regular model $\mathcal{M}$ found in Section III, we now address the second aspect of the problem, i.e., to give a function/procedure that returns a regularity score $R(s, j, \mathcal{M})$ for event $s[j]$, as well as the regularity $R(s, \mathcal{M})$ of the whole sequence s. The procedure we give is a novel "ping" message based algorithm that computes the regularity scores of all events in a sequence (at the same time), as well as the regularity score of the whole sequence.

The basic idea is that we let each event agent $a[i]$ in s initiate a ping message and send it to its right neighbor $a[i+1]$, which in turn propagates the message to its right neighbor, and so on. Then we see if we can match such an *(initiator,receiver)* pair to an edge e in $\mathcal{M}$. Intuitively, a match to an edge with endpoints lower in $\mathcal{T}$ is a better match, as the events are more specific. Thus, there is a multiplicative discount factor $\gamma (0 < \gamma \leq 1)$ for each level increment of one of the endpoins of *e* in $\mathcal{T}$ (γ may further be customized for particular nodes in $\mathcal{T}$, but we omit such details).

Informally, an irregular event is an event whose ping receives no response (in terms of matching an edge in $\mathcal{M}$), or who does not respond to anyone else' pings. This means that the event is "strange" in its context in the sequence.

Line 3 initializes two arrays $pre[i]$ and $post[i]$ that indicate the regularity of $s[i]$ with respect to the events before and after $s[i]$, respectively. As in BUILDREGULARMODEL, we iterate through each pair of event (message initiator and receiver, lines 5-7), and check if this event pair is an edge in the regular model (line 10). If it is, we get the weight of this edge (line 11). As discussed earlier, the weight w is based on a discount factor $\gamma (0 < \gamma \leq 1)$ that signifies the importance of an upper level edge. If both v_0 and v are leaves (i.e., level 0), then w is 1. If a model has fewer leaf level edges but more edges on higher level events, γ can be set higher to give more weight to such edges. We will get to this issue in the experiment section.

Lines 12-15 update the $pre[\cdot]$ value of the receiver and the $post[\cdot]$ value of the message initiator if we get a higher weight. In lines 16-17, the messages are passed on to the right neighbors. In the end, the regularity score $r[i]$ of each event $s[i]$ is the product of its $pre[\cdot]$ score and $post[\cdot]$ score (line 19), and the regularity score of the whole sequence is the average of the individual events' scores (line 20).

Since w, $pre[\cdot]$, and $post[\cdot]$ values are all between 0 and 1, the regularity scores are between 0 and 1 too. The lower the score, the more irregular the event (or sequence) is. This algorithm essentially does what we can do given the information of a set of sequences S: checking if the event in question has a context that is common in S (through its model $\mathcal{M}$). We find in the experiments (Section VII) that this

Algorithm 2: COMPUTEREGULARITY $(s, \mathcal{M})$

Input: s: an event sequence,
$\mathcal{M}$: regular event model
Output: regularity $r[i]$ of each event $s[i]$, and regularity r of s

```
for each event s[i] ∈ s in parallel do
    agent a[i] initiates a message and sends to a[i + 1]
    initialize pre[i] and post[i] to 0
for each time step t do
    for each agent a[i] in parallel do
        if a[i] receives a message msg then
            a[j] ← msg's initiator
            for each v0 ∈ ancestors (a[j]),
            v ∈ ancestors(a[i]) do
                if (v0, v) ∈ M then
                    w ← γ^(l(v0)+l(v))
                    if w > pre[i] then
                        pre[i] ← w
                    if w > post[j] then
                        post[j] ← w
    for each letter s[i] ∈ s in parallel do
        agent a[i] propagates its message to a[i + 1]
for each event s[i] ∈ s in parallel do
    r ← pre[i] · post[i]
r ← (Σ_{i=1}^{|s|} r[i]) / |s|
```

metric is very effective in identifying irregular events within a sequence, even though the sequence itself is overall regular (having a high regularity score). Note that an event at the very beginning (or very end, resp.) of a sequence may have the $pre[\cdot]$ score (or $post[\cdot]$ score, resp.) being 0, which gives a low regularity score. However, we can quickly find such an event at the border of a sequence and only look at its $post[\cdot]$ score (or $pre[\cdot]$ score, resp.) that truly indicates its regularity.

As another remark, consider the scenario that an event *e* is missing from a sequence. What impact does it have on the regularity scores of other events in this sequence? Informally, it depends on how "important" this event *e* is to other events (with respect to the regular model $\mathcal{M}$). If $\mathcal{M}$ is "linear" around *e*, like the one shown in Fig. 2(a), missing *e* would only affect *d*'s $post[\cdot]$ score and *f*'s $pre[\cdot]$ score — hence only *d* and *f*'s regularity scores. If, on the other hand, $\mathcal{M}$ is like in Fig. 2(b), where *e*'s incoming neighbor *d* has multiple outgoing edges, and *e*'s outgoing neighbor *f* has multiple incoming edges, then the regularity scores of these other events would not be significantly changed by the absence of *e*.

V. ANALYSIS AND OPTIMIZATION

Intuitively, during the BUILDREGULARMODEL algorithm, as a message propagates a long way, the chance that it will form new edges in G becomes very small. This is because it is more likely to have a "path" between two farther nodes as the number of possible paths increases. Hence, when the edge probability becomes low enough, we may stop the lengthy message propagation process. We now rigorously analyze this idea.

From an agent that initiates a message, let p_n denote the

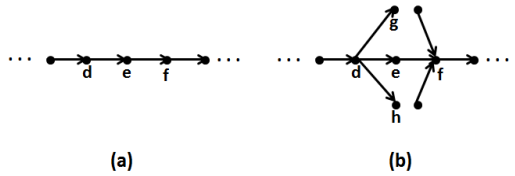

Fig. 2: Illustrating the impact of a missing event "e" from a sequence. (a) A regular model where missing *e* will only make events *d* and *f* irregular. (b) A regular model where missing *e* will not have a significant impact on the regularity scores of other events in the sequence.

probability that there exists an edge to an event n time steps later (i.e., an edge that spans *n* hops), and let q_n denote the probability that there exists a path to an event *n* time steps later. We devise a novel two function (p_n and q_n) recursive approach to solve p_n and q_n simultaneously.

We further define f_n as the probability that an event $s[i]$ in S and an event $s[i+n]$ is a frequent pair in S, i.e., the event order $s[i]$ followed by $s[i+n]$ appears in more than τ sequences in S. f_n can be estimated as the empirical probability as BUILDREGULARMODEL is run and messages are propagated incrementally.

We then have:

$$p_n = f_n \cdot (1 - \sum_{i=1}^{n-1} p_i q_{n-i}) \quad (1)$$

$$q_n = \sum_{i=1}^{n} p_i q_{n-i} \quad (2)$$

In Equation (1), there is an edge to a vertex *n* hops away (p_n) if and only if the pair is frequent (f_n) and there is no path with multiple edges between the two nodes ($1-\sum_{i=1}^{n-1} p_i q_{n-i}$). The case of multiple-edge path is subdivided according to the number of hops of the first edge in the path (i.e., p_i), and the remaining path (q_{n-i}). Similarly, in Equation (2), the case of existing a path of n hops is subdivided according to the number of hops of the first edge in the path (i.e., p_i). The boundary conditions are $q_0 = 1$ and $p_1 = f_1$.

It is not hard to see that, by applying Equations (2) and (1) alternately, we can iteratively obtain all q_i's and p_i's up to q_n and p_n (i.e., in the order q_1, p_2, q_2, p_3, and so on). We can stop the process, as well as the algorithm, when p_n is small enough (i.e., when *n* is big enough).

A few remarks are in place. The above analysis serves both as understanding of the model graph property (i.e., how likely an edge crosses over n events in the sequence), and as a performance optimization in that we can stop the algorithm early when p_n is small enough (i.e., the model is stable).

Secondly, we choose to empirically measure f_n and analytically derive p_n because f_n is much easier (in terms of accuracy) to be measured empirically (as a single frequency number for the whole set of sequences S). This is because typically $p_n \ll f_n$ when n is large, as evidenced by Equation (1) showing $p_n < f_n$ and shown empirically. It is well

known that a very small probability number p_n (when n is large) is extremely difficult to estimate empirically (requiring a very large sample size) [8]. Furthermore, the frequency measurement f_n is increasingly more accurate as time step progresses (n increases) since more event pairs have been seen so far as the basis for determining if the frequency threshold $\tau \cdot m$ is reached.

Our experiments in Section VII show that (1) p_n converges quickly to close to 0 (Fig. 14), (2) the performance improvement with early stop is dramatic, often more than an order of magnitude (Fig. 15), and (3) there is hardly any loss on accuracy, with precision 1 and recall 0.99 or higher (Fig. 16). The details are in Section VII. Thus, this optimization should always be performed.

The final remark is that this is a dynamic analysis and estimation on the fly based on statistics collected at execution time. It is impossible to do this statically, as the result depends on the actual data.

VI. Further Enhancement

Recall that, in the BUILDREGULARMODEL algorithm, we need to maintain a bitmap for each ordered pair (u, v) in S, which may be very space consuming when the number of events is large. Consider this general problem that is of independent interest:

> **Group Counting Problem:** *There are a large number of elements, and a set of m groups. There are continuous, possibly duplicate input pairs of (element, group) information. Use a succinct data structure to store this information. A query is of the following form: given an element, return an estimate of how many groups this element belongs to, and possibly which groups.*

When $m = 1$, this problem is exactly the commonly known *set membership problem*, and can be approximately solved by a hash table, or in particular, a Bloom filter [9]. Thus, the Group Counting problem is a generalization of the set membership problem.

The Group Counting problem is part of BUILDREGULARMODEL for counting the frequency of event pairs (lines 15-16 of BUILDREGULARMODEL). Specifically, each distinct event pair (v_1, v_2) is an element, and each sequence in S is a group (hence there are m groups). Given an event pair element, we need to query how many groups (sequences) it is in (and compare this number with $\tau \cdot m$). Therefore, we discuss our space-efficient approximate solution to the Group Counting problem.

We devise a novel approximation technique, which we call a *blended bitmap set* (BBS), to solve this problem. A BBS consists of k rows and w columns of bitmaps, while each bitmap has m bits. The choice of k and w is based on memory constraint and will be discussed shortly. We choose k hash functions $h_i (1 \le i \le k)$ randomly from a universal hash function family [10]. Each h_i corresponds to a row of w bitmaps. Fig. 3 shows an example BBS structure.

The basic idea is as follows. Whenever an element-group pair (e, g) arrives (meaning element e is in group g), we apply

	1	2	3	4	5
h_1 1	bm: 0110	bm: 0111		bm: 1000	
h_2 2			bm: 0111		bm: 1110
h_3 3			bm: 1000	bm: 0111	

Fig. 3: Illustrating a BBS structure where $k = 3$, $w = 5$, and $m = 4$. That is, there are 3 hash functions (one for each row), each hash function have 5 possible values (thus 5 columns), and there is a bitmap of 4 bits at each cell (for 4 groups). An empty cell indicates its bitmap is all 0's now.

each of the k hash functions h_i to e and get the hash value be $h_i(e)$. We take $\log w$ bits from $h_i(e)$, which corresponds to a column (say, column *j*) in the *i*th row. Then we set the group information (group g) in the bitmap of cell (i, j), denoted as $bm[i, j]$. However, we do not directly set bit g of $bm[i, j]$. Instead, we apply a random permutation $perm_i$ over the *m* bits, where $perm_i$ is also determined by $h_i(e)$ (by taking a few bits from it). Suppose bit g maps to bit g' with this permutation. Then we set bit g' of $bm[i, j]$. This is done for each of the k hash functions (k rows).

The query/lookup function on a BBS also first applies each of the k hash functions and locates cell (i, j) and $bm[i, j]$ in the same way. Then it applies the reverse of the permutation $perm_i^{-1}$ on $bm[i, j]$ (for each $1 \le i \le k$). This gets the original bitmaps back. Then we do a bitwise AND over the k recovered bitmaps to estimate the group membership and group count. The bit permutations in BBS operations are for balancing the collision probabilities at each bit position (robust to group size skewness) to accurately estimate group counts, which is detailed later. We show these two algorithms below.

Algorithm 3: BBS-SETELEMENTGROUP(bbs, e, g)

Input: bbs: a blended bitmap set,
e : element,
g : group id
Output: updated bbs, group size sum sum_0

```
1 for each i ← 1...k do
2     r_i ← h_i(e)
3     use bits from r_i to determine column id j ∈ 1...w
4     use bits from r_i to determine a random permutation perm_i
5     g' ← perm_i(g)
6     if bit g' of the bitmap bm[i,j] in cell c[i,j] = 0 then
7         set that bit to 1
8 if line 7 is performed at least once then
9     sum_0 ← sum_0 + 1
```

BBS-SETELEMENTGROUP is exactly as discussed above. The r_i in line 2 is the (random) hash value from the ith hash function. One detail we have not discussed is the permutation $perm_i$ in line 4. A fully uniformly random permutation over m bits requires $\log m!$ bits to describe. This is too many bits to

take from r_i. Moreover, doing $perm_i$ and $perm_i^{-1}$ would be rather costly. As mentioned earlier, the goal of permutation is to have an accurate estimation of group count (detailed later). We must balance between feasibility and accuracy. Thus, we partition a bitmap into a constant number b of blocks, and permute the blocks uniformly at random. This only requires $\log b!$ bits from r_i, and is much more efficient. There are standard algorithms to do the random permutation of b blocks, such as the Knuth shuffle [11] with a cost $O(b)$.

Line 5 is to apply this permutation function to bit position g, and g' is the position it is permuted to (along with its block). We set that bit (line 7). Lines 8-9 are to maintain a counter sum_0, which is the total (estimated) number of distinct elements currently in the BBS — it can be an under-estimate due to collisions. This will be used in BBS-LOOKUP for adjusting the group count estimate.

Algorithm 4: BBS-LOOKUP(bbs, e, sum_0)

Input: bbs: a blended bitmap set,
e : element,
sum_0: current group size sum
Output: bitmap of groups e is in, and estimated group count

1 $gm \leftarrow$ bitmap of all 1's // *group membership of* e
2 **for** ***each*** $i \leftarrow 1 \dots k$ **do**
3 $r_i \leftarrow h_i(e)$
4 use bits from r_i to determine column id $j \in 1 \dots w$
5 use bits from r_i to determine a random permutation $perm_i$
6 $bm' \leftarrow perm_i^{-1}(bm[i,j])$
7 $gm \leftarrow gm \& bm'$
8 $c_0 \leftarrow sum_0/m$
9 **if** c_0 *has changed significantly from previous call* **then**
10 $i \leftarrow 0$; $p_0 \leftarrow 0$
11 **repeat**
12 $p_{i+1} \leftarrow [1 - (1 - \frac{1}{w})^{\frac{c_0}{1-p_i}}]^k$
13 $i \leftarrow i + 1$
14 **until** p_i *converges; let the converged value be* p
15 $x \leftarrow$ number of 1's in gm
16 **return** gm and $\frac{x-mp}{1-p}$

Line 1 of BBS-LOOKUP initializes a bitmap gm to be all 1's for its m bits. Lines 2-4 are as before to apply each of the k hash functions and get to cell (i, j) of BBS. Then line 6 applies the reverse permutation to $bm[i, j]$, i.e., the reverse operations of the b block permutation for BBS-SETELEMENTGROUP. Let the recovered bitmap be bm' (line 6). Note that only the same element e will use the same $perm_i$ as it is determined by bits of $h_i(e)$. Line 7 is the bitwise AND operation. Thus, after the loop, gm ends up being the bitwise AND of the k original bitmaps (before permutation) corresponding to e. Clearly, the bits corresponding to all the groups that element e is in must be 1 in gm.

At this point, although gm contains the estimated group membership of element e, the total group count can be an overestimate due to collisions on other bits.

Theorem 1: After the loop in lines 2-7 of BBS-LOOKUP, any bit in gm that should be 0 (for a group that e is not in) has an estimated false positive probability of $[1 - (1 - \frac{1}{w})^c]^k$, where c is the average group size.

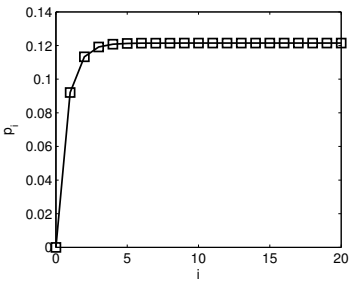

Fig. 4: The convergence of the p_i series

Proof: Element e maps to k cells in BBS (one in each row) due to the hash functions. For this bit in gm (let it be bit x) to be a false positive, it must have been set to 1 in the bitmaps of all these k cells. Let us consider these k cells one at a time, say cell (i, j). In order for bit x of $bm[i, j]$ to be 1, there are two conditions: (1) with a random permutation determined by its hash value, another elements group ID landed on bit x; and (2) that element's h_i hash value happens to go to j. The probability of (2) is $\frac{1}{w}$, and due to random permutation, in expectation, c distinct elements satisfy (1), where c is the average group size stated in the theorem, i.e., the average number of distinct elements in each group. Therefore, the probability that at least one of such collisions happens in cell (i, j) is $1 - (1 - \frac{1}{w})^c$. For bit x to be a false positive, this needs to be true for all k cells, giving the probability stated in the theorem. ■

In line 8 of BBS-LOOKUP, c_0 is an under-estimate of the average group size c due to collisions (false positives).

Theorem 2: The p_i series in the loop at lines 10-14 will converge. Moreover, the converged value p is an unbiased estimate of the bit false positive probability in gm, and $\frac{x-mp}{1-p}$ is an unbiased estimate of the number of groups that e is in.

Proof: First, we claim that the p_i series in line 12 monotonically increases. We prove this by induction on i. In the base case, when $i = 0$, clearly $p_{i+1} > p_i$ since $p_1 = [1-(1-\frac{1}{w})^{c_0}]^k > 0 = p_0$. For the induction step, suppose $p_i > p_{i-1}$ is true. Then we show that $p_{i+1} > p_i$. This is true because $[1 - (1 - \frac{1}{w})^{\frac{c_0}{1-x}}]^k$ increases when x increases from p_{i-1} to p_i. Since the p_i series monotonically increases, and since it has an upper bound 1, from the Monotone Convergence Theorem [12], it must converge. Furthermore, since $\frac{c_0}{1-p}$ is an unbiased estimate of c, the converged value p must also be an unbiased estimate of the bit false positive probability from Theorem 1. Finally, if the true total number of groups that e is in is y, then $x = y + (m - y)p$ in expectation, which gives $y = \frac{x-mp}{1-p}$ as an unbiased estimation. ■

Fig. 4 plots the convergence of the p_i series with (arbitrary) parameters $k = 3$, $w = 500$, and $c_0 = 300$. Finally, let us discuss the optimal choice of parameters k and w. Suppose we are given a space budget. Since each cell of BBS contains a bitmap of the same size, it is equivalent to having an upper bound on $k \cdot w$. Let this limit be $k \cdot w = M$.

Theorem 3: Given the constraint $k \cdot w = M$, the choice of k and w that minimizes the bit false positive probability in Theorem 1 is $k = \frac{M}{c} \ln 2$ and $w = \frac{c}{\ln 2}$.

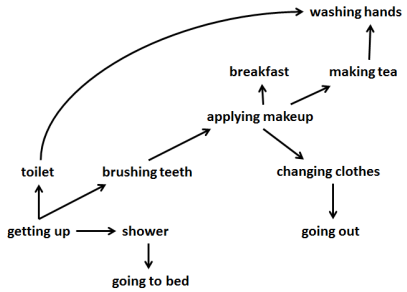

Fig. 5: A snippet (subgraph) of the regular model learned.

Proof: From Theorem 1, the false positive probability is $[1-(1-\frac{1}{w})^c]^k \cong (1-e^{-\frac{c}{w}})^k$. To minimize it, we use Lagrange multipliers [13] and define a function

$$\Lambda(k, w, \lambda) = (1 - e^{-\frac{c}{w}})^k + \lambda(kw - M)$$

where we use the constraint that $kw = M$. We then solve the equation system $\frac{\partial \Lambda}{\partial k} = 0$, $\frac{\partial \Lambda}{\partial w} = 0$, and $\frac{\partial \Lambda}{\partial \lambda} = 0$, which gives $k = \frac{M}{c} \ln 2$ and $w = \frac{c}{\ln 2}$ as in the theorem. ■

Note that c is variant as new (element, group) pairs are inserted. Thus, there is no optimal fixed choice throughout the process. In our problem, we estimate the maximum number of event pairs separately for each sequence (e.g., by sampling), and make the parameter choice optimal for the time when BBS nearly reaches its maximum element count.

VII. Experiments

In this section, we perform a systematic evaluation of our work using two real world datasets and a hybrid dataset (combining user input data with simulation tools). We study the effectiveness and efficiency of our framework and approach. Specifically, we answer the following questions:

- How effective is our regular model? In particular, what are some properties of the model graphs such as vertex degrees? Do the models make sense?
- What irregular events can we find? If we "inject" some irregular events, can they be found using the model? How does it compare with previous work?
- The model building algorithm based on message passing is easily parallelizable. Can we verify this observation even in the most commonplace computing environment — multicore computers as for most laptops and desktops today?
- Recall that we estimate the new-edge probability in a model graph by an analysis based on a dynamic measurement on the fly. What does this probability series look like in practice? Is it effective in early stopping w.r.t. performance improvement and model accuracy?
- How effective is the BBS optimization in the tradeoff between memory consumption and model accuracy?

A. Datasets and Setup

We use the following real-world and hybrid datasets to perform the empirical study:

Smart house data. This dataset originates from the work of Tapia et al. [14], downloaded from [15]. Between 77 and 84 sensor data collection boards equipped with reed switch sensors were installed in two single-person apartments collecting data about human activity for two weeks. The sensors were installed in everyday objects such as drawers, refrigerators, containers, etc. to record opening-closing events (activation deactivation events) as the subject carried out everyday activities. Activities associated with the sensor signals are labeled in the data.

Hybrid data. This is based on the research work on human activity recognition [16]. The dataset is called *hybrid* in that it is designed to combine user input data on daily activities and sensor signals with simulation tools for flexibility. We use it to test our algorithms on data with various parameters.

GPS data. This GPS trajectory dataset was collected in Microsoft Research's Geolife project over five years [17], [18]. A GPS trajectory is represented by a sequence of timestamped points, each of which contains the information of latitude, longitude, height, speed and heading direction. This dataset contains 17,621 trajectories with a total distance of 1,292,951 kilometers and a total duration of 50,176 hours. These trajectories were recorded by different GPS loggers and GPS-phones. This dataset is 1.6 GB.

We implement all the algorithms in the paper — both a serial version and a multi-threaded version — in Oracle Java 1.8.0_45. In addition, we download the Machine Learning software Weka [7] to use its Cobweb conceptual clustering implementation for obtaining event hierarchies. All experiments are performed on a machine with an Intel Core i7 2860QM (Quad-Core) 2.50 GHz processor and an 8GB memory.

In an excellent survey on anomaly detection for sequences [3], Chandola et al. categorize all previous work into solving three problems. Even though it is not specifically designed for event sequences as in our solution, potentially applicable to our problem is their Category 2, i.e., detecting short subsequences in a long sequence T that are anomalous w.r.t. the rest of T (the other two categories are finding whole sequences that are anomalous w.r.t. a sequence database, and determining if a given query pattern in a sequence is anomalous w.r.t. its expected frequency, resp.). To apply it to our problem, a concatenation of the sequences over all time units is T, and the detected short subsequences are irregular events. We implement the t-STIDE based algorithm described in [3] for Category 2 problems, which is also an overall best algorithm as found in the experimental study by Chandola et al. [19]. We compare with this work on the effectiveness in finding irregular events.

B. Experimental Results

We build a regular model for each dataset. The regular models of the smart house data and the hybrid data indicate the inherent regularity in daily activity sequences. For GPS data, there is a concentration area at latitude range [39.90725, 40.05447] and longitude range [116.31632, 116.51534] (an area in Beijing near Microsoft Research Asia). We divide this rectangle into a 32×32 grid, where each of the 1024 areas is about $0.27km^2$, and is treated as a discrete location. Thus, our model signifies the regularity in the location and movement

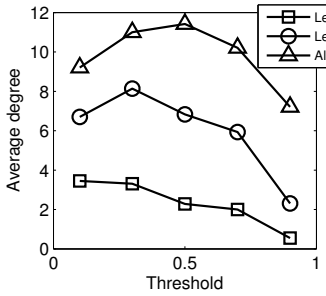

Fig 6: Vertex degrees (Hybrid)

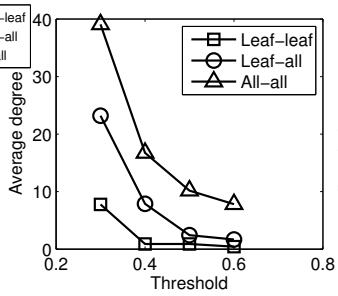

Fig 7: Vertex degrees (GPS)

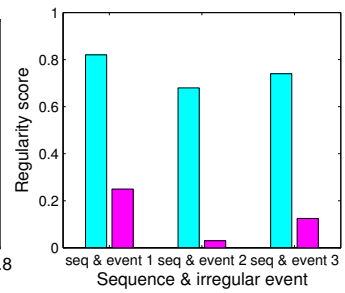

Fig 8: Irregular events (SH)

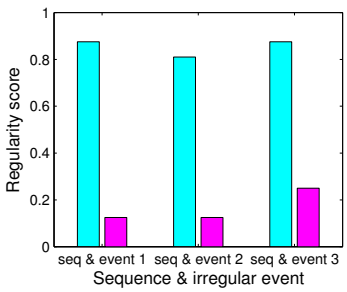

Fig 9: Irregular events (Hybrid)

daily event sequence of a user. From these regular models, we can identify irregular events.

Models and Finding Irregular Events. We first examine the effectiveness of our models. All the regular models for the three datasets indicate the event context and order information as described in Section III. Varieties and generalizations of events are also encoded in the model through event hierarchies. Let us take a look at an example snippet of a model.

Fig. 5 shows a subgraph of the model learned from the hybrid data (similar to the smart house data). As discussed earlier, the model graph shows the *preceding* and *succeeding context* of events allowing possible interleaving events between two events connected by an edge. Moreover, the model is succinct. For instance, there is no edge between "getting up" and "applying makeup", since a path *getting up* $\rightarrow$ *brushing teeth* $\rightarrow$ *applying makeup* is already formed when the "message" from "getting up" reaches "applying makeup" in the BUILDREGULARMODEL algorithm.

For clarity, we only show some "leaf to leaf" edges in Fig. 5. Our regular model also includes event vertices at various levels of the event hierarchy produced by conceptual clustering (Cobweb). Hence there are also edges between vertices of higher levels. For example, in the model of Fig. 5, leaf nodes "making tea" and "making coffee" have a parent node in the event hierarchy, indicating the generalization relationship. Let us label this parent node "making beverage". There is an edge from "applying makeup" to "making beverage" which is not shown in Fig. 5. However, there is no edge from "applying makeup" to "making coffee", since the person mostly drinks tea in the morning. The edge from leaf to a higher level node allows variation and generalization of events, and there is some"penalty" in this generalization when computing the regularity score in COMPUTEREGULARITY (line 10). If the person one day makes coffee instead of tea, "making beverage" will receive a smaller regularity score — unusual but not completely impossible.

To have a better understanding of the model graph property, we measure various types of node degrees — the average leaf to leaf degree, leaf to any node, and any node to any node degrees. Fig. 6 shows the result of these statistics for the model of the hybrid dataset, for various threshold parameter τ values in BUILDREGULARMODEL. We can see that the average leaf-leaf degree ranges between 4 and 0, and decreases as τ increases. This is because as the frequency threshold increases, fewer event pairs are qualified for edges. The average leaf to any node (marked leaf-all in Fig. 6) and the average all-all degrees are higher as they are for more general event types. An interesting phenomenon here is that, when τ decreases in the low value range, the average node degrees may actually decrease. This is because, when τ is smaller, paths between two nodes tend to be formed earlier, preventing the creation of an edge between these two (more distant) nodes. Hence the curve is not necessarily monotonic.

The result on the smart house dataset is very similar and is omitted. We show the result for the GPS dataset in Fig. 7. While the trend is roughly similar, the leaf-leaf degree is lower when τ is above 0.4 for the GPS data than for the hybrid data. This indicates that the user behavior is less regular at the leaf level, which corresponds to fine-granularity of location ($0.27km^2$ grids). The behavior is more regular when the location grids are coarser, at higher levels of the event hierarchy. Such edges are useful too, and the weights (in line 10 of COMPUTEREGULARITY) can be adjusted to penalize higher level edges less. Any leaf-leaf edge has a weight 1, and the weight is multiplied by a discount factor $\gamma \leq 1$ when an endpoint of the edge increases a level in the hierarchy. By default, we use $\gamma = 0.5$ for the first two datasets and $\gamma = 0.8$ for the GPS dataset.

We next examine the ability of our regular model in finding irregular events. For each dataset, we can find a number of unusual events — activities or places visited. We show a few of them here. Fig. 8 lists a few from the smart house data. The first group of two bars in Fig. 8 is the regularity scores of a sequence and an irregular event $e1$ therein, respectively. This irregular event $e1$ is "Preparing lunch", and its context contains "going to work" both in the morning (before $e1$) and in the afternoon (after $e1$). We find that in this dataset the user rarely prepares lunch at home if there are events "going to work" both in the morning and afternoon (presumably the user usually eats lunch at work). The regularity score of $e1$ is around 0.2, much lower than the score of the whole sequence (around 0.8).

To test the model's ability in locating irregular events, we also inject two events $e2$ and $e3$ which have unusual contexts in the dataset. $e2$ is"watching TV" and the context is in the morning, while $e3$ is "preparing snack" and the context is at night. We insert $e2$ and $e3$ into two random sequences respectively, build the model, and report the regularity scores. The second group of two bars in Fig. 8 show the scores of the sequence and $e2$, and the third group similarly for $e3$.

Fig. 9 shows the result for the hybrid dataset. The first group of two bars is for an existing event $e1$ "making tea",

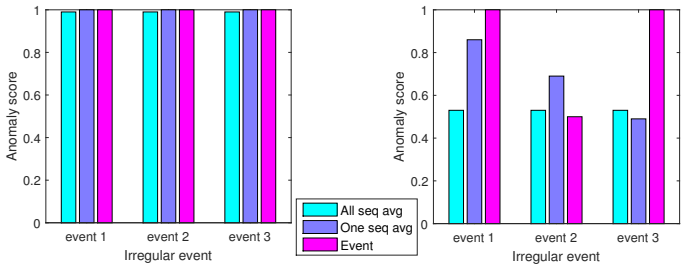

Fig 10: Previous work (SH data)

Fig 11: Previous work (Hybrid)

and the context does not have "heating water" anywhere before it (but the sequence is merely "shower", "applying makeup", "making tea", followed by"making breakfast"). In the dataset, however, making tea almost always has heating water somewhere in front of it. The second group of two bars is for another existing event $e2$ "having breakfast", but there is no "making breakfast" prior to it, which gives $e2$ a low regularity score. The third group of two bars is for an injected event, where we change an existing event "making tea" to a new event $e3$ "making coffee". As discussed earlier, this user rarely makes coffee in the morning, but both leaf nodes generalize to the parent event node "making beverage". The regularity score of the original "making tea" is 1, while the new $e3$ "making coffee" only has a score of 0.25, which is slightly higher than $e1$ and $e2$ due to the match of edges over the parent node "making beverage".

Then we examine the effectiveness of using the t-STIDE algorithm in previous work for Category 2 problem [3], as discussed at the end of Sec. VII-A. We set the window-size parameter k to be 8, as found optimal in [19]. Fig. 10 shows the result with the smarthouse data for the same three irregular events as in Fig. 8, and Fig. 11 with the hybrid data for the same events in Fig. 9. Unlike our approach, this line of work shows the *anomaly score*—the higher the score is (closer to 1), the more anomalous it is. For each irregular event, we show three bars, with the first being the average anomaly score over all sequences, the second being the average of the sequence where the event is in, and the third being the anomaly score of the event itself. We can see that this algorithm is completely ineffective in locating the three irregular events in smarthouse data; all bars show anomaly scores close to 1. For hybrid data (Fig. 11), it is better with event 1, but the sequence average is still close to $e1$'s score, indicating a great number of false positives inside the sequence. The detection result is completely wrong with $e2$, as its anomaly score is even lower than the sequence average. The result for $e3$ is fine after changing "making tea" to "making coffee"; however, we find that before the change, when $e3$ is "making tea", its score is still close to 1, which is erroneous as it should not be an anomaly. When we change the parameter k to smaller values, all scores decrease, but it is still very ineffective in isolating the target irregular events.

The reason is that the previous approaches are mostly designed for time series data with its discretized version [3]. They do not work well for our problem because relevant context events, which mostly depend on time, can be rather far away in terms of the number of other events in between. The other events in between may also be in any order (especially with the smarthouse data). This causes a serious problem for previous work. Our approach based on message passing and graph edges is much more robust to such *noisy* event sequences, since messages will be able to reach the relevant context events.

Performance and Optimizations. In the next set of experiments, we examine the efficiency of the algorithm for building regular models. We have observed that our message passing algorithmic framework in BUILDREGULARMODEL can be easily parallelized. This is indeed the case in our implementation. We can easily write a multi-threaded program that runs on the most commonplace hardware — multicore processors. Our experimental machine has *four cores*. We show the execution time results on the GPS dataset over various threshold τ values in Fig. 12. As τ increases, execution time decreases. This is because a greater τ implies that fewer event pairs are qualified as edges in the model graph, and hence the bookkeeping work as well as "reachability" check in BUILDREGULARMODEL is reduced. Furthermore, we can see that the multi-threaded version achieves nearly four times speedup over the serial version under the quad-core machine, verifying the parallelizability of the algorithm.

In Fig. 13, likewise, we show the execution time result for the hybrid dataset, varying the period over which a regular model is learned. The period ranges from 45 days to 730 days (typically it is not desirable to learn a model over too long a period, as model changes over time, and should be re-learned). There is a very interesting phenomenon here: as period increases, the speedup achieved by the multi-threaded version is much more than 4 times (its execution time still increases with the period, but much slower). This is because, as the number of events to be processed is large, the multi-threaded version starts from many positions in the sequences, and finds edges and paths in the model graph much earlier than the serial version. This has positive feedback ("the rich gets richer"), in that such edges and paths will prune much work on those event pairs immediately. Certainly in principle a more careful implementation of the serial version may improve its performance to be only 4 times worse than the multi-threaded version; the point is that such an implementation may involve heavy tuning and is hard to do.

We next empirically verify our analysis in Section V of the probability of a new edge as message passing progresses over time. The optimization here is that the algorithm can stop early when this estimated probability on the fly is small enough. The only statistics we need to collect during BUILDREGULARMODEL is f_n for time step n, the empirical probability that an event pair encountered at this time step is frequent (above threshold τ). Then our algorithm in Section V will estimate the probability of a new edge at this time step. We plot this probability series over time steps in Fig. 14, for all three datasets. After a short initial instability, the probability of new edge quickly converges to almost 0 within 5 to 8 time steps for all three datasets. The initial instability is due to inaccurate estimate of f_n in the beginning, as the number of total event pairs encountered is still small. Since f_n is more accurate over time, the confidence of our estimation of the probability of new edge also increases over time. Fig. 14 informs us that the early stop optimization should be very effective, as the model should usually be very stable within a limited number of time steps.

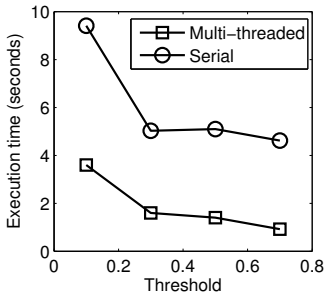

Fig 12: Execution time (GPS)

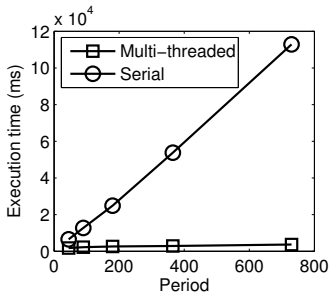

Fig 13: Execution time (Hybrid)

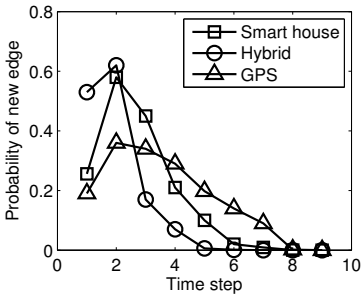

Fig 14: New edge probability

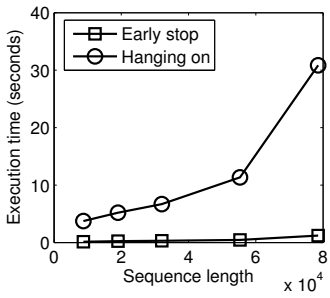

Fig 15: Early stop optimization

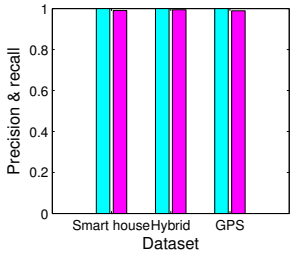

Fig 16: Accuracy of early stop

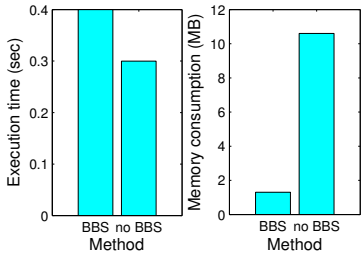

Fig 17: BBS optimization

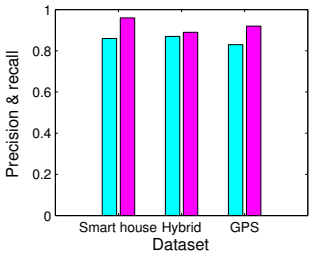

Fig 18: Accuracy of BBS optim.

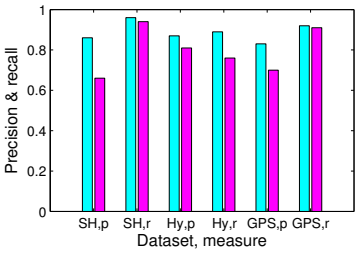

Fig 19: Group count adjustment

This is indeed verified in our next two figures, on performance and accuracy (of the resulting model), respectively.

Using the 90 days hybrid dataset, varying the average daily sequence length, in Fig. 15, we show the execution times with and without the early stop optimization (i.e., stop when the probability of new edge is below 0.01), marked as "early stop" and "hanging on" in the figure, respectively. We can see that the performance improves tremendously (easily more than an order of magnitude), especially for longer sequences, since stopping after a small number of time steps saves more work.

We then examine the accuracy of the resulting model after this optimization, shown in Fig. 16 for all three datasets. We show the *precision* and *recall* of the edges in the model graph with the optimization, compared to the one without optimization. Precision is defined as the fraction of edges in the model under optimization that are also in the one without optimization, while recall is the other way around. We can see that in all three datasets, the model under optimization is almost identical to the original one. In particular, the precisions are all 1.0, and the recalls are 0.99 or higher. Precision is always 1.0 is exactly expected, because the execution with early stop optimization is identical to the original one before stopping, and hence all the edges it gets in the model are also in the original model. In summary, this set of experiments (Figures 14-16) gives us more confidence that the early stop optimization should always be used.

In the final set of experiments, we examine the BBS optimization. We first show the performance result in Fig. 17 for the hybrid dataset over a period of 365 days, measuring both execution times (left plot) and memory consumption (right plot), where we explicitly set the memory constraint of the BBS version to be about 1/8 of the original execution (which accordingly determines the optimal parameters of BBS). While significantly saving memory space, one tradeoff of the BBS optimization is the slight increase of execution time, as shown in the left plot. In both plots, the left bar is with BBS optimization and the right bar is without. The slight increase in execution time is due to applying a constant number of universal hashing and block random permutation during BBS operations.

In Fig. 18, we show the accuracy of the resulting model in terms of precision and recall (defined in the same way as for the early stop optimization). We can see that the precision and recall are typically around 0.9 or more, indicating that the obtained model is quite accurate. We verify that they can still be used to find the irregular events discussed earlier.

Finally, we evaluate and verify the necessity of the group count adjustment in lines 8-16 of BBS-LOOKUP, rather than simply returning line 15 as the count estimate. In Fig. 19, we show the precision and recall changes without this adjustment for all three datasets, where SH (smart house), Hy (hybrid), and GPS stands for the three datasets, while "p" and "r" stands for precision and recall, respectively. In each group of two bars, the left one is the original BBS, and the right one is without the count adjustment. This experiment shows that the adjustment is needed for better accuracy. Overall, BBS optimization is useful in computing environments where memory is more of a concern.

Summary of Results. The empirical study in this section is enlightening. The experiments give us more concrete understanding of the regular models we find, as well as their graph properties (node degrees). The regularity scores combined with the models are very effective and informative in finding irregular events and sequences. The model building algorithm using message passing framework is efficient and can be easily parallelized. Our analysis and dynamic estimation of probability of new edge on the fly are very effective — the resulting early stop optimization should always be used.

Finally, the BBS optimization is effective and useful when reducing memory consumption is a concern.

VIII. Related Work

Outlier/anomaly detection has been studied for sequence data (cf. a survey [3]), temporal data (cf. a survey [4]), and in more general contexts (cf. a survey [5]). Most related to our work is the one for sequence data [3], in which Chandola et al. categorize all previous work into solving three problems. Closest to our work in the second category, i.e., detecting short subsequences in a long sequence T that are anomalous w.r.t. the rest of T, while the other two categories are finding whole sequences that are anomalous w.r.t. a sequence database, and determining if a given known query pattern in a sequence is anomalous w.r.t. its expected frequency. Since our main goal is to find irregular events and we can concatenate the sequences across all time units into one (huge) sequence, previous work in this category (detecting anomalous subsequences) could potentially be used in the problem we study. However, as shown in Section VII, it does not work well because relevant context events, which mostly depend on time, can be rather far away in terms of the number of other events in between. The other events in between may also be in any order. This causes a serious problem for previous work. Our approach based on message passing and graph edges is much more robust to such *noisy* event sequences, since messages will be able to reach the relevant context events.

Matsubara et al. [20] study the problem of segmentation and regime detection in correlated time series occurring at the same time. One example they give is to detect the dancing steps from multiple signals of a dance. We are not concerned with segmentation and regime detection, as we only have known time units in our problem (and applications). Event detection and activity recognition in general has been studied in the ubiquitous computing and artificial intelligence communities (e.g., [2]). This line of work, however, is not concerned with building a regular model over activities/events from multiple time units and identifying irregular events. Our work is built on top of such event detection work, as we build our regular model (as well as event hierarchy) on top of discrete events.

Complex event processing has been extensively studied in the data management literature (e.g., [21] as a survey). It focuses on detecting the occurrence of a particular complex event pattern, but not about finding a regular model or irregular events. Previous work on Bloom filters [9] and count min sketches [22] bear some similarity with our blended bitmap set (BBS) technique. However, BBS is a non-trivial extension to solve the Group Counting problem that we identify. For example, a count min sketch only has counters at each cell of the hash table, while we need to use bitmaps to represent a potentially large number of groups. Moreover, we use random permutations to overcome the skewness of group sizes, and dynamically estimate the average group size and collision probability to adjust group counts. Finally, Beedkar and Gemulla [23] study frequent sequence mining under item hierarchies. This is related to our event hierarchy concept. However, they study the classic frequent itemset mining problem under item hierarchies. Thus, it is a different problem and they have no intention to learn a regular model from sequences over many time units or to find irregular events.

IX. Conclusions

In this paper, we study a novel problem with practical significance. We propose an algorithmic method based on easily parallelizable message passing framework to build a regular model graph, as well as a method to compute regularity scores. One of our optimizations is to dynamically estimate new edge probability based on runtime statistics and speeds up model building often by over an order of magnitude, while the other significantly saves space consumption. Our systematic experimental study verifies the effectiveness and efficiency of our approaches.

Acknowledgment

This work was supported in part by the NSF, under the grants IIS-1239176 and IIS-1319600.

References

[1] http://www.chicagotribune.com/news/nationworld/ct-germanwings-pilot-controlled-descent-20150506-story.html.

[2] T. Plötz, N. Y. Hammerla, and P. Olivier, "Feature learning for activity recognition in ubiquitous computing," in *IJCAI*, 2011.

[3] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection for discrete sequences: A survey," *TKDE*, 2012.

[4] M. Gupta, J. Gao, C. Aggarwal, and J. Han, "Outlier detection for temporal data," *TKDE*, 2014.

[5] V. J. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, 2004.

[6] D. H. Fisher, "Knowledge acquisition via incremental conceptual clustering," *Machine learning*, 1987.

[7] http://sourceforge.net/projects/weka/files/weka-3-7-windows-x64/.

[8] A. Mood, F. Graybill, and D. Boes, "Introduction to the theory of statistics. 3rd mcgraw-hill," 1974.

[9] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *CACM*, 1970.

[10] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," in *STOC*, 1977.

[11] D. E. Knuth, "Seminumerical algorithms, the art of computer programming, vol. 2," 1981.

[12] W. A. Sutherland, *Introduction to metric and topological spaces*. Oxford University Press, 1975.

[13] G. B. Arfken, *Mathematical methods for physicists*. Academic press, 2013.

[14] E. M. Tapia, S. S. Intille, and K. Larson, *Activity recognition in the home using simple and ubiquitous sensors*. Springer, 2004.

[15] http://courses.media.mit.edu/2004fall/mas622j/04.projects/home/.

[16] G. Azkune, A. Almeida, D. López-de Ipiña, and L. Chen, "Combining users' activity survey and simulators to evaluate human activity recognition systems," *Sensors*, vol. 15, no. 4, pp. 8192–8213, 2015.

[17] Y. Zheng, L. Liu, L. Wang, and X. Xie, "Learning transportation mode from raw gps data for geographic applications on the web," in *WWW*, 2008.

[18] http://research.microsoft.com/apps/pubs/?id=141896.

[19] V. Chandola, V. Mithal, and V. Kumar, "Comparative evaluation of anomaly detection techniques for sequence data," in *ICDM*, 2008.

[20] Y. Matsubara, Y. Sakurai, and C. Faloutsos, "Autoplait: Automatic mining of co-evolving time sequences," in *SIGMOD*, 2014.

[21] G. Cugola and A. Margara, "Processing flows of information: From data stream to complex event processing," *ACM Computing Surveys*, 2012.

[22] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Foundations and Trends in Databases*, vol. 4, no. 1–3, pp. 1–294, 2012.

[23] K. Beedkar and R. Gemulla, "Lash: Large-scale sequence mining with hierarchies," in *SIGMOD*, 2015.