This article has been accepted for publication in IEEE Journal on Selected Areas in Communications. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2022.3191123

An Effective Machine Learning based Algorithm for Inferring User Activities from IoT Device Events

Guoliang Xue, *Fellow, IEEE,* Yinxin Wan, *Student Member, IEEE,* Xuanli Lin, *Student Member, IEEE,* Kuai Xu, *Senior Member, IEEE,* Feng Wang, *Member, IEEE*

***Abstract*—The rapid and ubiquitous deployment of Internet of Things (IoT) in smart homes has created unprecedented opportunities to automatically extract environmental knowledge, awareness, and intelligence. Many existing studies have adopted either machine learning approaches or deterministic approaches to infer IoT device events and/or user activities from network traffic in smart homes. In this paper, we study the problem of inferring user activity patterns from a sequence of device events by first deterministically extracting a small number of representative user activity patterns from the sequence of device events, then applying unsupervised learning to compute an optimal subset of these user activity patterns to infer user activities. Based on extensive experiments with sequences of device events triggered by $2,959$ real user activities and up to $30,000$ synthetic user activities, we demonstrate that our scheme is resilient to device malfunctions and transient failures/delays, and outperforms the state-of-the-art solution.**

***Index Terms*—Machine learning, IoT device events, user activities.**

1. Introduction

The ubiquitous and heterogeneous deployment of Internet of Things (IoT) devices in smart homes has created new opportunities to extract knowledge, awareness, and intelligence via monitoring and understanding the devices' interactions with their environments and users [4, 6, 7, 9, 10, 13, 19, 20]. A number of studies focused on discovering meaningful information from network traffic of IoT devices in smart homes, even though much of the traffic is often encrypted over secure wireless networks or via IoT application-level encryption. For example, [2] and [11] adopted well-known machine learning (ML) models to infer device events from network packets, while PingPong [15] and IoTAthena [17] utilized deterministic algorithms for device event extraction based on the observation that every device event generates a repeatable sequence of network packets.

In addition, several recent studies have explored the feasibility of inferring user activities with IoT devices in smart homes. For example, [12] utilized an unsupervised learning method to discover user activities based on information collected by sensors deployed in a smart home. The study in [1] demonstrated the possibility of passively sniffing wireless-only network traffic of IoT devices for detecting and identifying IoT device types and user activities in smart homes from an adversary perspective. [18] introduced the problem of User Activity Inference (UAI) and proposed an approximate matching-based algorithm to infer a multiset of user activities from the network traffic of IoT devices, which was actively collected at programmable home routers from a trusted home user perspective. However, reconstructing real-world user activities in smart homes and matching them with the ground-truth requires an exact *sequence* of user activities, rather than a *multiset* of user activities produced by IoTMosaic [18].

Towards this end, in this paper, we study the problems of Events to Activities (`E2A`) and Events to Activity Patterns (`E2AP`) from a trusted home user perspective for inferring *a sequence of user activities* from a sequence of IoT device events, which can be extracted from network traffic using existing methods such as PingPong [15] and IoTAthena [17]. We design a two-phase scheme employing both deterministic algorithms and machine learning techniques for solving the `E2AP` and `E2A` problems.

In Phase 1, we compute a small number of *representative* matches of the activity patterns. We prove that the solution computed based on these representative matches is as good as any solution that can be obtained by considering all possible matches. In Phase 2, we present an efficient algorithm for computing a compatible set of matches of user activity patterns with maximum total weight, for any given weight assignment to the matches. We then design an unsupervised learning algorithm for computing a good set of weights. While the loss function for our problem is non-differentiable, we design an exact algorithm for minimizing the loss function over any one of the weight variables. Similar to many ML algorithms that build optimization models and learn optimal parameters [14], we design a coordinate descent algorithm for solving `E2AP` which converges in a finite number of iterations.

The main contributions of this paper are the following:

- We formulate the problem of inferring a sequence of user activities and their patterns in smart homes from a sequence of device events extracted from network traffic.
- We prove that one can concentrate on a small number of representative matches of user activity patterns and design an efficient algorithm for computing these representatives.
- We design an efficient algorithm to compute a compatible set of matches of user activity patterns with maximum total weight for any given weight assignment. We then design an algorithm to compute an optimal set of weights in a finite number of iterations via unsupervised learning.
- We conduct extensive experiments with both real and synthetic data and demonstrate that our algorithm is robust and outperforms the state-of-the-art solution.

The rest of this paper is organized as follows. Section 2 formulates the problems `E2A` and `E2AP` and presents some

All authors are affiliated with Arizona State University. Emails: {xue, ywan28, xlin54, kuai.xu, fwang25}@asu.edu. This research was supported in part by NSF grants 2007469, 1816995, and 1717197. The information reported here does not reflect the position or the policy of the funding agency.

basic concepts needed in the paper. Section 3 presents Phase 1 of our scheme, while Sections 4 and 5 present Phase 2 of our scheme. Section 6 presents evaluation results. Section 7 concludes the paper. Proofs are presented in the appendix.

2. Problem Formulation and Basics

Recent research [1, 18] suggests that a user activity in a smart home may be inferred from the sequence of IoT device events triggered by the activity. For example, one of the user activities studied in [18] *"A person without key entering the home from the front door during the day"* usually triggers the sequence of 10 IoT device events (*Ring doorbell motion detection, Ring spotlight motion detection, Ring doorbell ringing, Ring doorbell stream on, Ring doorbell stream off, August lock manual unlocking, Tessan contact sensor open, Tessan contact sensor close, August lock manual locking, Arlo Q camera motion detection*), involving 5 devices. Intuitively, observing this sequence of 10 device events in a very short period of time suggests that the above user activity very likely has happened.

If a device malfunctions when a user activity occurs, the device events corresponding to this device will be missing from the sequence. For example, if the Tessan contact sensor malfunctions, the events *Tessan contact sensor open* and *Tessan contact sensor close* will be missing in the corresponding sequence of device events. If the Ring doorbell is in sleeping mode when the user activity occurs, the event *Ring doorbell motion detection* will be delayed. Because the amount of delay varies significantly in our experiments, we consider such device event as missing as well.

We have a *set* $\mathcal{A} = \{A^1, A^2, \ldots, A^{|\mathcal{A}|}\}$ of *user activities*. Each occurrence of a user activity $A \in \mathcal{A}$ triggers a sequence of IoT *device events* $\mathbb{S}^1(A) = (e_1^{A,1}, e_2^{A,1}, \ldots, e_{|\mathbb{S}^1(A)|}^{A,1})$ or a subsequence of $\mathbb{S}^1(A)$, where the missing events correspond to devices that malfunction when A occurs. We use $|\mathcal{A}|$ to denote the *cardinality* of set $\mathcal{A}$ and use $|\mathbb{S}|$ to denote the *length* of sequence $\mathbb{S}$. We use $\mathbb{A} = (A_1, A_2, \ldots, A_{|\mathbb{A}|})$ to denote a *sequence* of user activities. We use *superscripts* in the description of (distinct) user activities in *set* $\mathcal{A}$, and use *subscripts* in the description of (not necessarily distinct) user activities in *sequence* $\mathbb{A}$. A^1 and A^2 denote two distinct elements in set $\mathcal{A}$. A_1 and A_2 denote the first and second elements in sequence $\mathbb{A}$, respectively.

For a user activity A, we use $\mathcal{S}(A) = \{\mathbb{S}^1(A), \mathbb{S}^2(A), \ldots, \mathbb{S}^{|\mathcal{S}(A)|}(A)\}$ to denote the set of distinct sequences of device events that could be triggered by A, where $\mathbb{S}^k(A) = (e_1^{A,k}, e_2^{A,k}, \ldots, e_{|\mathbb{S}^k(A)|}^{A,k})$ for $k = 1, 2, \ldots, |\mathcal{S}(A)|$, and $\mathbb{S}^k(A)$ is a subsequence of $\mathbb{S}^1(A)$ for $k > 1$. We call $\mathcal{S}(A)$ the *set of possible patterns of* A. In our experiments, $\mathcal{S}(A)$ is extracted through observing the device event sequences triggered by repeated occurrences of A.

Each IoT device may correspond to multiple events. Each user may correspond to multiple user activities, and two different user activities may both involve a common device event. *The focus of this paper is on user activities* (rather than users) and *device events* (rather than devices).

We use several examples to illustrate important concepts and algorithms throughout the paper. For ease of understanding, we use the following setting for all of our examples.

Example Setting: $\mathbb{E} = (a, b, a, b, d, c, a, b, c)$ is the sequence of device events. $\mathcal{A} = \{A^1, A^2\}$ is the set of user activities. The set of possible patterns for A^1 (A^2, respectively) is $\mathcal{S}(A^1) = \{(a, b)\}$ ($\mathcal{S}(A^2) = \{(a, b, c), (a, b)\}$, respectively).

The sequence of device events triggered by a sequence $\mathbb{A}$ of user activities, denoted by $E(\mathbb{A})$, satisfies

$$E(\mathbb{A}) \in \{Z_1 \| Z_2 \| \cdots \| Z_{|\mathbb{A}|} \mid Z_j \in \mathcal{S}(A_j), \forall j = 1, \ldots, |\mathbb{A}|\}. \quad (1)$$

Example 1: Let $\mathbb{A}^1 = (A^1, A^2, A^2)$ and $\mathbb{A}^2 = (A^2, A^2, A^2)$ be two sequences of user activities, where A^1 and A^2 are as in the example setting. Then $E(\mathbb{A}^1)$ could be any of (a, b, a, b, c, a, b, c), $(a, b, a, b, c, \quad a, b)$, (a, b, a, b, a, b, c), or (a, b, a, b, a, b). Similarly, $E(\mathbb{A}^2)$ could be any of $(a, b, c, a, b, c, a, b, c)$, (a, b, c, a, b, c, a, b), (a, b, c, a, b, a, b, c), (a, b, c, a, b, a, b), (a, b, a, b, c, a, b, c), (a, b, a, b, c, a, b), (a, b, a, b, a, b, c), or (a, b, a, b, a, b). □

This example shows that the same sequence of user activities may trigger different sequences of device events, while different sequences of user activities may trigger the same sequence of device events.

The *nondeterministic* mapping E maps a sequence of user activities to a sequence of device events. We are interested in the reverse problem: *inferring the sequence of user activities from a given sequence of IoT device events*. We call it the `E2A` problem (*Events to Activities*), defined in the following.

Problem 1 (`E2A`): Let $\mathcal{A} = \{A^1, A^2, \ldots, A^{|\mathcal{A}|}\}$ be a set of user activities. For each $A \in \mathcal{A}$, its set of possible patterns $\mathcal{S}(A)$ is known. Given a sequence of IoT device events $\mathbb{E} = (e_1, e_2, \ldots, e_m)$, the `E2A` problem seeks for a sequence $\mathbb{A} = (A_1, A_2, \ldots, A_{|\mathbb{A}|})$ of user activities in $\mathcal{A}$ that is most likely to trigger the sequence of device events $\mathbb{E}$. □

We further propose the *Events to Activity Patterns* problem (`E2AP`) to infer a sequence of user activities together with their patterns. The solution to this problem can be used to provide accurate quantification of the solution quality.

Problem 2 (`E2AP`): Let $\mathcal{A} = \{A^1, A^2, \ldots, A^{|\mathcal{A}|}\}$ be a set of user activities. For each $A \in \mathcal{A}$, its set of possible patterns $\mathcal{S}(A)$ is known. Given a sequence of IoT device events $\mathbb{E} = (e_1, e_2, \ldots, e_m)$, the `E2AP` problem seeks for a sequence $\mathbb{A} = (A_1, A_2, \ldots, A_{|\mathbb{A}|})$ of user activities in $\mathcal{A}$ together with pattern $\mathbb{S}_j^{k_j} \in \mathcal{S}(A_j)$ for $j = 1, 2, \ldots, |\mathbb{A}|$, such that $\mathbb{S}_1^{k_1} \| \mathbb{S}_2^{k_2} \| \cdots \| \mathbb{S}_{|\mathbb{A}|}^{k_{|\mathbb{A}|}}$ is as close to $\mathbb{E}$ as possible. □

A solution to the `E2AP` problem consists of a sequence $\mathbb{A} = (A_1, A_2, \ldots, A_{|\mathbb{A}|})$ of user activities and a corresponding sequence $\mathbb{S} = (\mathbb{S}_1^{k_1}, \mathbb{S}_2^{k_2}, \ldots, \mathbb{S}_{|\mathbb{A}|}^{k_{|\mathbb{A}|}})$ of activity patterns. We can use $\mathbb{A}$ as a solution to the `E2A` problem. The edit distance [5] between $\mathbb{E}$ and $\mathbb{S}_1^{k_1} \| \mathbb{S}_2^{k_2} \| \cdots \| \mathbb{S}_{|\mathbb{A}|}^{k_{|\mathbb{A}|}}$ can be used as a metric to quantify the solution quality.

The `E2AP` problem is closely related to `UAI` problem studied in [18]. While both [18] and this paper have the same ultimate goal of inferring user activities in a smart home, the specific goals and techniques used are very different. `UAI` produces a *multiset of user activities*; `E2AP` produces a *sequence of user activity patterns*. Note that we can use a sequence of user activity patterns to produce a sequence of user activities which in turn can be used to produce a multiset of user activities, but not *vice versa*. A sequence of user activity patterns can be directly compared with the

sequence of device events to measure the quality of a solution for E2AP. Such comparison is not possible if the output is a multiset or a sequence of user activities (without the patterns). In UAI, each user activity $A \in \mathcal{A}$ has one *signature*, which is the full sequence of device events that may be triggered by A (corresponding to $\mathbb{S}^1(A)$ in this paper), but notes that it may trigger a subsequence of the signature. When computing the matching of a signature, [18] relies on k-approximate subsequence matching, and gives high priorities to the full sequence of the signature over a partial sequence. In E2AP, we consider all possible signature patterns independently, and decide the weights for the patterns using unsupervised learning, where we aim to minimize the edit distance between the given sequence of device events and the concatenation of the sequence of computed activity patterns. This makes our ML-based scheme resilient to device malfunctions and transient failures/delays, a significant advantage over the algorithm of [18].

We design a two-phase scheme for solving E2AP. Phase 1 computes a small number of representative matches in $\mathbb{E}$ for each possible pattern $\mathbb{S}^k(A^i)$ of each user activity $A^i \in \mathcal{A}$. We prove that the solution computed based on these small number of representative matches is as good as any solution that can be obtained by considering all matches. Phase 2 consists of Phase 2A and Phase 2B. Phase 2A computes a compatible sequence $\mathbb{S}^w$ of representative matches of user activity patterns with maximum total weight, for any given weight assignment w of the matches. The edit distance between $\mathbb{E}$ and the concatenation of the user activity patterns in $\mathbb{S}^w$ is the value of the loss function $f(w)$. Phase 2B aims to compute an optimal weight w via unsupervised learning. Both Phase 1 and Phase 2B are executed once. Phase 2A is executed multiple times, one for each evaluation of $f(w)$ within Phase 2B.

Table 1 describes notations commonly used in this paper.

An important concept is the match of a possible activity pattern in the sequence of device events. We study these in the rest of this section.

Definition 1 (match and minimal match): Let $\mathbb{E} = (e_1, e_2, \ldots, e_m)$ be the sequence of device events. Let $A \in \mathcal{A}$ be a user activity with $\mathcal{S}(A) = \{\mathbb{S}^1(A), \mathbb{S}^2(A), \ldots, \mathbb{S}^{|\mathcal{S}(A)|}(A)\}$ as its set of possible patterns, where $\mathbb{S}^k(A) = (e_1^{A,k}, e_2^{A,k}, \ldots, e_{|\mathbb{S}^k(A)|}^{A,k})$ for $k = 1, 2, \ldots, |\mathcal{S}(A)|$. A *match* of $\mathbb{S}^k(A)$ in $\mathbb{E}$, denoted by $\psi_i^{A,k}$, $i \in \mathbb{Z}^+$, is a sequence of positive integers $(\psi_{i,1}^{A,k}, \psi_{i,2}^{A,k}, \ldots, \psi_{i,|\mathbb{S}^k(A)|}^{A,k})$ such that

$$1 \le \psi_{i,j}^{A,k} < \psi_{i,j'}^{A,k} \le m, \forall 1 \le j < j' \le |\mathbb{S}^k(A)|, \quad (2)$$

$$e_{\psi_{i,j}^{A,k}} = e_j^{A,k}, \forall 1 \le j \le |\mathbb{S}^k(A)|. \quad (3)$$

We call $[\psi_{i,1}^{A,k}, \psi_{i,|\mathbb{S}^k(A)|}^{A,k}]$ the *interval* of $\psi_i^{A,k}$, and denote it as $\psi_i^{A,k}.interval$. We use $\Psi^{A,k}$ to denote the set of all matches of $\mathbb{S}^k(A)$ in $\mathbb{E}$. A match $\psi_i^{A,k}$ of $\mathbb{S}^k(A)$ in $\mathbb{E}$ is called *minimal*, if there is no match $\psi_{i'}^{A,k}$ of $\mathbb{S}^k(A)$ in $\mathbb{E}$ whose interval is a proper subset of the interval of $\psi_i^{A,k}$. We use $\Psi^{min,A,k}$ to denote the set of all minimal matches of $\mathbb{S}^k(A)$ in $\mathbb{E}$. □

Example 2: Set A to A^2 in the example setting. Then $\mathbb{S}^1(A) = (a, b, c)$ has 9 matches in $\mathbb{E}$: $\psi_1^{A,1} = (1, 2, 6)$, $\psi_2^{A,1} = (1, 2, 9)$, $\psi_3^{A,1} = (1, 4, 6)$, $\psi_4^{A,1} = (1, 4, 9)$, $\psi_5^{A,1} = (1, 8, 9)$, $\psi_6^{A,1} = (3, 4, 6)$, $\psi_7^{A,1} = (3, 4, 9)$, $\psi_8^{A,1} = (3, 8, 9)$, $\psi_9^{A,1} = (7, 8, 9)$. Among the 9 matches in $\Psi^{A,1}$, $\psi_1^{A,1}$, $\psi_3^{A,1}$, $\psi_6^{A,1}$, $\psi_9^{A,1}$ are minimal. $\mathbb{S}^2(A)$ has 6 matches in $\mathbb{E}$: $\psi_1^{A,2} = (1, 2)$, $\psi_2^{A,2} = (1, 4)$, $\psi_3^{A,2} = (1, 8)$, $\psi_4^{A,2} = (3, 4)$, $\psi_5^{A,2} = (3, 8)$, $\psi_6^{A,2} = (7, 8)$. $\psi_1^{A,2}$, $\psi_4^{A,2}$, $\psi_6^{A,2}$ are minimal. □

TABLE 1
COMMONLY USED NOTATIONS IN THIS PAPER

Notations	Description
$\mathcal{A}$	set of user activities: $\mathcal{A} = \{A^1, A^2, \ldots, A^{\lvert\mathcal{A}\rvert}\}$
A^i	the ith user activity: $A^i \in \mathcal{A}$
A or A_j	a user activity: $A, A_j \in \mathcal{A}$
$\mathbb{A}$	sequence of user activities: $\mathbb{A} = (A_1, A_2, \ldots, A_{\lvert\mathbb{A}\rvert})$
$\mathbb{E}$	sequence of IoT device events: $\mathbb{E} = (e_1, e_2, \ldots, e_m)$
$\mathbb{E}(i:j)$	portion of $\mathbb{E}$ from index i to index j
$\mathcal{S}(A)$	set of distinct device event sequences triggered by $A \in \mathcal{A}$: $\mathcal{S} = \{\mathbb{S}^1(A), \mathbb{S}^2(A), \ldots, \mathbb{S}^{\lvert\mathcal{S}(A)\rvert}(A)\}$
$\mathbb{S}^k(A)$	the k-th possible device event sequence triggered by $A \in \mathcal{A}$: $\mathbb{S}^k(A) = (e_1^{A,k}, e_2^{A,k}, \ldots, e_{\lvert\mathbb{S}^k(A)\rvert}^{A,k})$
$\psi_i^{A,k}$	a match of $\mathbb{S}^k(A)$ in $\mathbb{E}$, with two interchangeable descriptions in the paper: $\psi_i^{A,k} = (\psi_{i,1}^{A,k}, \psi_{i,2}^{A,k}, \ldots, \psi_{i,\lvert\mathbb{S}^k(A)\rvert}^{A,k})$ or $\psi_i^{A,k} = (\psi_i^{A,k}[1], \psi_i^{A,k}[2], \ldots, \psi_i^{A,k}[\lvert\mathbb{S}^k(A)\rvert])$
$\phi_i^{A,k}$	similar to $\psi_i^{A,k}$
$\Psi^{A,k}$	set of all matches of $\mathbb{S}^k(A)$ in $\mathbb{E}$
$\Psi^{min,A,k}$	set of all minimal matches of $\mathbb{S}^k(A)$ in $\mathbb{E}$
Ψ	union of $\Psi^{A,k}$ over all combinations of $A \in \mathcal{A}$ and $k \in \{1, 2, \ldots, \lvert\mathcal{S}(A)\rvert\}$
Ψ^{min}	union of $\Psi^{min,A,k}$ over all combinations of $A \in \mathcal{A}$ and $k \in \{1, 2, \ldots, \lvert\mathcal{S}(A)\rvert\}$
$\mathbb{L}^{min,A,k}$	list of (not necessarily all) minimal matches of $\mathbb{S}^k(A)$ in $\mathbb{E}$: whose i-th node may be denoted as $\mathbb{L}^{min,A,k}[i]$
$\mathbb{L}^{min}$	2-D array of lists of minimal matches of $\mathbb{S}^k(A)$ in $\mathbb{E}$ for all possible combinations of $A^i \in \mathcal{A}$ and $k \in \{1, 2, \ldots, \lvert\mathbb{S}^k(A)\rvert\}$: indexed as $\mathbb{L}^{min}[i,k] = \mathbb{L}^{min,A^i,k}$
n_i	number of elements in $\mathbb{S}(A^i)$, $i = 1, 2, \ldots, \lvert\mathcal{A}\rvert$
N	$N = \sum_{i=1}^{\lvert\mathcal{A}} n_i$
$m_{i,k}$	$m_{i,k} = \lvert\mathbb{L}^{min,A^i,k}\rvert$, $i = 1, \ldots, \lvert\mathcal{A}\rvert$, $k = 1, \ldots, \lvert\mathbb{S}^k(A^i)\rvert$
M	$M = \sum_{i=1}^{\lvert\mathcal{A}\rvert} \sum_{k=1}^{\lvert\mathbb{S}^k(A^i)\rvert} m_{i,k}$
$\mathbb{M}$	1-D array of 4-tuple (A, k, α, β): $\mathbb{M}[i]$, $i = 1, 2, \ldots, M$
$w(i,k)$	weight for $\mathbb{S}^k(A^i)$, $i = 1, \ldots, \lvert\mathcal{A}\rvert$, $k = 1, \ldots, \lvert\mathcal{S}(A^i)\rvert$
$\lVert$	concatenation operator
$\mathbb{Z}^+$	set of nonnegative real numbers

In order to design efficient algorithms for solving the E2AP problem, we restrict our attention to $O(m)$ matches of $\mathbb{S}^k(A)$ in $\mathbb{E}$ for each A and k, without losing important information. To achieve this goal, we introduce the following concepts.

Definition 2 (precedence relation on $\Psi^{A,k}$): We define a binary relation $\preceq$ on $\Psi^{A,k}$ as follows. Let $\psi_i^{A,k}$ and $\psi_{i'}^{A,k}$ be two elements in $\Psi^{A,k}$. We say $\psi_i^{A,k} \preceq \psi_{i'}^{A,k}$ if

(i) $\psi_{i,|\mathbb{S}^k(A)|}^{A,k} < \psi_{i',|\mathbb{S}^k(A)|}^{A,k}$, or

(ii) $\psi_{i,|\mathbb{S}^k(A)|}^{A,k} = \psi_{i',|\mathbb{S}^k(A)|}^{A,k}$ and $(\psi_{i,|\mathbb{S}^k(A)|-1}^{A,k}, \psi_{i,|\mathbb{S}^k(A)|-2}^{A,k} \cdots, \psi_{i,1}^{A,k})$ is lexicographically greater than or equal to $(\psi_{i',|\mathbb{S}^k(A)|-1}^{A,k}, \psi_{i',|\mathbb{S}^k(A)|-2}^{A,k} \cdots, \psi_{i',1}^{A,k})$.

When $\phi \preceq \psi$ and $\phi \neq \psi$, we say ϕ *precedes* ψ, denoted by $\phi \prec \psi$. We say that ϕ is *lighter than* ψ when $\phi \prec \psi$. □

One can verify that $\preceq$ defined above is a *linear ordering* [16] on $\Psi^{A,k}$ (as well as on $\Psi^{min,A,k}$). In other words, we have

1) For any $\phi, \psi \in \Psi^{A,k}$, at least one in $\{\phi \preceq \psi,\ \psi \preceq \phi\}$ is true.
2) If $\phi \preceq \psi$ and $\psi \preceq \phi$, then $\phi = \psi$.
3) If $\phi \preceq \psi$ and $\psi \preceq \gamma$, then $\phi \preceq \gamma$.

Definition 3 (equivalence relation on $\Psi^{A,k}$ and $\Psi^{min,A,k}$): Let $A \in \mathcal{A}$, and $\mathbb{S}^k(A)$ be a possible pattern of A. Let $\psi_i^{A,k}$

and $\psi_{i'}^{A,k}$ be two elements in $\Psi^{A,k}$ ($\Psi^{min,A,k}$, respectively). We say that $\psi_i^{A,k}$ is *equivalent* to $\psi_{i'}^{A,k}$, denoted by $\psi_i^{A,k} \equiv \psi_{i'}^{A,k}$, if the intervals of $\psi_i^{A,k}$ and $\psi_{i'}^{A,k}$ are the same. □

Clearly, $\equiv$ is a binary relation defined on $\Psi^{A,k}$, as well as a binary relation defined on $\Psi^{min,A,k}$. One can verify that the binary relation $\equiv$ defined on $\Psi^{A,k}$ ($\Psi^{min,A,k}$, respectively) is an *equivalence relation* [16]. This equivalence relation partitions the set $\Psi^{A,k}$ ($\Psi^{min,A,k}$, respectively) into *equivalence classes* where two matches in $\Psi^{A,k}$ ($\Psi^{min,A,k}$, respectively) are in the same equivalence class if and only if they are equivalent.

Definition 4 (representative matches): Let $A \in \mathcal{A}$, and $\mathbb{S}^k(A)$ be a possible pattern of A. For each equivalence class of $\Psi^{A,k}$ ($\Psi^{min,A,k}$, respectively), we choose the lightest element in the class as its *representative*. □

Example 3: Set A to A^2 in the example setting. There are 9 members in $\Psi^{A,1}$. In lexicographically order, they are $(1,2,6)$, $(1,2,9)$, $(1,4,6)$, $(1,4,9)$, $(1,8,9)$, $(3,4,6)$, $(3,4,9)$, $(3,8,9)$, $(7,8,9)$. In lightest first order, we have $(3,4,6) \prec (1,4,6) \prec (1,2,6) \prec (7,8,9) \prec (3,8,9) \prec (1,8,9) \prec (3,4,9) \prec (1,4,9) \prec (1,2,9)$.

The 5 equivalence classes of $\Psi^{A,1}$, with the representative of each class listed first in bold font, are $\{(\mathbf{3},\mathbf{4},\mathbf{6})\}$, $\{(\mathbf{1},\mathbf{4},\mathbf{6}),(1,2,6)\}$, $\{(\mathbf{7},\mathbf{8},\mathbf{9})\}$, $\{(\mathbf{3},\mathbf{8},\mathbf{9}),(3,4,9)\}$, and $\{(\mathbf{1},\mathbf{8},\mathbf{9}),(1,4,9),(1,2,9)\}$. The 2 equivalence classes of $\Psi^{min,A,1}$ are $\{(\mathbf{3},\mathbf{4},\mathbf{6})\}$ and $\{(\mathbf{7},\mathbf{8},\mathbf{9})\}$. □

In Example 3, each equivalence class of $\Psi^{min,A,k}$ has only one match. In general, the number of elements in an equivalence class of $\Psi^{min,A,k}$ ($\Psi^{A,k}$) may be very large.

Lemma 1: Let $A \in \mathcal{A}$, and $\mathbb{S}^k(A)$ be a possible pattern of A. The number of equivalence classes of $\Psi^{A,k}$ is no more than $m(m+1)/2$. The number of equivalence classes of $\Psi^{min,A,k}$ is no more than m, where $m = |\mathbb{E}|$. □

3. Computing Representatives Matches (Phase 1)

In this section, we present Phase 1 of our two-phase scheme as outlined in the previous section. We first present Algorithm 1 which can be used to compute a sequence $\mathbb{L}^{min,A,k}$ of the representatives for the equivalence classes of $\Psi^{min,A,k}$, for any given $A \in \mathcal{A}$, and any $k \in \{0,1,\ldots,|\mathcal{S}(A)|$.

The data structures used in Algorithm 1 are as follows.

- $c[]$ is an integer-valued 2-D array of $m+1$ rows and $|\mathbb{S}^k(A)|+1$ columns. The entry $c[i,j]$, when computed, is equal to $|\text{LCS}(\mathbb{E}(start : i), \mathbb{S}^k(A)(1 : j))|$. Here $\text{LCS}(X,Y)$ denotes a *longest common subsequence* of X and Y, and $|Z|$ denotes the length of sequence Z.
- Integer l is used to index the next match $\psi_l \in \Psi^{min,A,k}$ found. At the end of the algorithm, l is the number of equivalence classes of $\Psi^{min,A,k}$.
- $\mathbb{L}^{min,A,k}$ is a singly linked list, each node of which is an array of $|\mathbb{S}^k(A)|$ integers, corresponding to the $|\mathbb{S}^k(A)|$ indices of a match in $\Psi^{min,A,k}$. The operation $\mathbb{L}^{min,A,k}.append$ appends a new node/match at the end.

The following is the flow of Algorithm 1. Lines 1 and 4 perform initialization. In particular, we initialize $c[start-1,j]$ to 0, which is $|\text{LCS}(\mathbf{null}, \mathbb{S}^k(A)(1 : j))|$ for all j and all $start$. We also initialize $c[i,0]$ to 0, which is $|\text{LCS}(\mathbb{E}(start : i), \mathbf{null})|$ for all i and all $start$.

Algorithm 1: AkMatch($\mathbb{E}, A, k$)

Input: Sequence of device events $\mathbb{E} = (e_1, e_2, \ldots, e_m)$, user activity A, and integer $k \in \{1, |\mathcal{S}(A)|\}$.
Output: List $\mathbb{L}^{min,A,k}$ of all representatives of equivalence classes of $\Psi^{min,A,k}$, in increasing order w.r.t $\prec$.

```
1  η ← |S^k(A)|; L^{min,A,k} ← null; l ← 0; start ← 1; i ← 1;
   /* finding a match of S^k(A) in E(start:m) */
2  for j := 0 to η do c[start − 1, j] ← 0;
3  while i ≤ m do
4      c[i, 0] ← 0;
5      for j := 1 to η do
6          if e_i = e_j^{A,k} then
7              c[i, j] ← c[i − 1, j − 1] + 1;
8          else if c[i − 1, j] ≥ c[i, j − 1] then
9              c[i, j] ← c[i − 1, j];
10         else c[i, j] ← c[i, j − 1];
11     if c[i, η] = n then
           /* a new match of S^k(A) is found */
12         l ← l + 1; initialize ψ_l; row ← i; col ← η;
13         while col > 0 do
14             if e_row = e_col^{A,k} then
15                 ψ_l[col] ← row; row ← row − 1; col ← col − 1;
16             else if c[row − 1, col] ≥ c[row, col − 1] then
17                 row ← row − 1;
18             else col ← col − 1;
19         L^{min,A,k}.append(ψ_l);
20         start ← ψ_l[1] + 1; i ← start; goto 2;
21     i ← i + 1;
22 output L^{min,A,k}.
```

In Line 1, we set both $start$ and i to 1 and get ready to compute the lightest minimal match of $\mathbb{S}^k(A)$ in $\mathbb{E}$ among those whose interval is contained in $[start, m]$. Line 20 also sets new values of $start$ and i before control goes to Line 2 to start the computation of the next match in $\Psi^{min,A,k}$.

In consecutive executions of the **while** loop in Line 3 with the same $start$ value, we aim to compute the lightest minimal match of $\mathbb{S}^k(A)$ in $\mathbb{E}$ among those whose intervals are contained in $[start, m]$, for the corresponding $start$ value. Note that $start$ is initialized to 1 in Line 1, and updated to a new (larger) value $\psi_l[1]+1$ in Line 20 after the l-th match is found. This update guarantees that the next match computed will be different from any of the previously computed matches.

For each fixed value of i, we compute $c[i,j]$ for $j = 1,2,\ldots,n$ in the **for** loop of Line 5. The condition in Line 6 is true if and only if $e_i = e_j^{A,k}$, indicating that we just found a matched pair of events. Due to this match, we have $|\text{LCS}(\mathbb{E}(start : i), \mathbb{S}^k(A)(1 : j))| = 1 + |\text{LCS}(\mathbb{E}(start : i-1), \mathbb{S}^k(A)(1 : j-1))|$. Therefore in Line 7, we set $c[i,j] \leftarrow c[i-1, j-1] + 1 = |\text{LCS}(\mathbb{E}(start : i-1), \mathbb{S}^k(A)(1 : j-1))| + 1 = |\text{LCS}(\mathbb{E}(start : i), \mathbb{S}^k(A)(1 : j))|$. Otherwise, $c[i,j]$ is either set to $c[i-1,j]$ in Line 9 or $c[i,j-1]$ in Line 10 to guarantee $c[i,j] = |\text{LCS}(\mathbb{E}(start : i), \mathbb{S}^k(A)(1 : j))|$.

In Line 11, we check whether a match of $\mathbb{S}^k(A)$ in $\mathbb{E}(start : i)$ is found. If this condition is not true, control jumps to Line 21 where i is incremented and the statement of the **while** loop is executed again if the new value of i does not exceed m, and the algorithm outputs $\mathbb{L}^{min,A,k}$ if the new value of i exceeds m. If the condition in Line 11 is true, we

backtrack to trace out the newly found match of $\mathbb{S}^k(A)$ in $\mathbb{E}(start : i)$ in Lines 12 to 19.

In Line 20, we update the values of $start$ and i to $\psi_l[1]+1$ and $\psi_l[1]$, respectively. In Line 2, we initialize the entries of $c[]$ in row $start-1$ to 0 to prepare for the computation of the lightest minimal match of $\mathbb{S}^k(A)$ in $\mathbb{E}(start : m)$. The previous values in the overwritten rows are no longer needed.

Example 4: We use the example setting to illustrate the execution of AkMatch($\mathbb{E}, A^2, 1$), with the aid of Table 2.

TABLE 2
ILLUSTRATION OF AKMATCH($\mathbb{E}, A^2, 1$)

			0	1 a	2 b	3 c
	0		0	0	0	0
$start_1$	1	a	0	1	1	1
	2	b	0	1	2	2
	3	a	0,0	0, 1	0,2	0,2
$start_2$	4	b	0,0	0,1	1, 2	1,2
	5	d	0,0	0,1	1,2	1,2
	6	c	0,0	0,1	1,2	2, 3
	7	a	0,0	0, 1	0,1	0,2
$start_3$	8	b	0,0	0,1	1, 2	1,2
	9	c	0,0	0,1	1,2	2, 3

Pattern $\mathbb{S}^1(A^2) = (a, b, c)$ is on top, and sequence $\mathbb{E} = (a, b, a, b, d, c, a, b, c)$ is on the left. To make things clear, we use $start_1$, $start_2$, and $start_3$ to denote the different start values. We use black font for $c[i, j]$ entries computed with $start_1 = 1$; red for $c[i, j]$ entries computed with $start_2 = 4$; and blue for $c[i, j]$ entries computed with $start_3 = 8$. For an entry that is written more than once, we use comma(s) to separate these values, with newer values towards the left.

First, we set $start$ to 1 (denoted by $start_1$) and initialize $c[0, j]$ to 0 for $0 \le j \le 3$. For $i = 1$, we have $c[1, 0] = 0$, $c[1, 1] = 1$ (since $e_1 = e_1^{A^2,1} = a$), $c[1, 2] = 1$, $c[1, 3] = 1$. Similarly, we have $c[2, 0]=0$, $c[2, 1]=1$; $c[2, 2]=2$, $c[2, 3]=2$; $c[3, 0] = 0$, $c[3, 1] = 1$; $c[3, 2] = 2$, $c[3, 3] = 2$; $c[4, 0] = 0$, $c[4, 1] = 1$; $c[4, 2] = 2$, $c[4, 3] = 2$; $c[5, 0] = 0$, $c[5, 1] = 1$; $c[5, 2] = 2$, $c[5, 3] = 2$; $c[6, 0] = 0$, $c[6, 1] = 1$; $c[6, 2] = 2$, $c[6, 3]=3$. Now we have $c[i, 3] = 3$, with $i = 6$. This tells us that there is a match of $\mathbb{S}^1(A^2)$ in $\mathbb{E}$ that lies within $\mathbb{E}(1 : 6)$.

The algorithm backtracks from cell $[6, 3]$ to trace out the lightest match of $\mathbb{S}^1(A^2)$ in $\mathbb{E}$ that lies within $\mathbb{E}(1 : 6)$. We shade the cells on the backtracking path where we use green background and a frame-box around the value of $c[row, col]$ if the condition in Line 14 is true, and use gray background if the condition in Line 16 is true. In cell $[6, 3]$, we set $\psi_1[3] \leftarrow 6$, and move to cell $[5, 2]$. In cell $[5, 2]$, we move to cell $[4, 2]$. In cell $[4, 2]$, we set $\psi_1[2] \leftarrow 4$, and move to cell $[3, 1]$. In cell $[3, 1]$, we set $\psi_1[1] \leftarrow 3$, and move to cell $[2, 0]$. This backtrack stops at cell $[2, 0]$. By now, we have found $\psi_1^{min,A^2,1} = (3, 4, 6)$. *Note that there may be multiple matches of* $\mathbb{S}^1(A^2)$ *in* $\mathbb{E}$ *that lie entirely in* $\mathbb{E}(1 : 6)$. *The match* $(3, 4, 6)$ *computed by our algorithm is minimal and the lightest among these matches*. In our example, $(1, 4, 6)$ and $(1, 2, 6)$ also lie within $\mathbb{E}(1 : 6)$. As illustrated in Example 3, $(3, 4, 6)$ is minimal and is lighter than both $(1, 4, 6)$ and $(1, 2, 6)$.

Next, we set $start \leftarrow \psi_1^{min,A^2,1}[1] + 1 = 4$, and start the computation of the next match. Since some of the cells will be overwritten by the algorithm, we use red font for the values computed with this new start value. We find $c[9, 3] = 3$, and backtrack to find the next match $\psi_2^{min,A^2,1} = (7, 8, 9)$, which is the only match of $\mathbb{S}^1(A^2)$ in $\mathbb{E}$ that lies entirely in $\mathbb{E}(4 : 9)$.

Then, we set $start \leftarrow \psi_2^{min,A^2,1}[1] + 1 = 8$, and start the computation of the next match. The cell values computed in this round are in blue font. This time, the algorithm could not find a match. In summary, the algorithm finds two matches of $\mathbb{S}^1(A^2)$ in $\mathbb{E}$: $\psi_1^{min,A^2,1} = (3, 4, 6)$ and $\psi_2^{min,A^2,1} = (7, 8, 9)$. □

Theorem 1: For given $\mathbb{E}$, A, and k, Algorithm 1 computes list $\mathbb{L}^{min,A,k}$ of all representatives of equivalence classes of $\Psi^{min,A,k}$, in increasing order defined by ordering $\prec$. The worst-case running time of the algorithm is $O(m^2|\mathbb{S}^k(A)|)$.□

Each match in $\Psi^{A,k}$ can be used as a possible match of the pattern $\mathbb{S}^k(A)$ of user activity A. In order to avoid *double-counting* of a device event, we introduce the following concept.

Definition 5 (compatible matches): Let $\psi_i^{A,k}$ be a match in $\Psi^{A,k}$ for $A \in \mathcal{A}$ and $k \in \{1, |\mathcal{S}(A)|\}$. Let $\psi_{i'}^{A',k'}$ be a match in $\Psi^{A',k'}$ for $A' \in \mathcal{A}$ and $k' \in \{1, |\mathcal{S}(A')|\}$. We say that $\psi_i^{A,k}$ and $\psi_{i'}^{A',k'}$ are *compatible* if the intervals of $\psi_i^{A,k}$ and $\psi_{i'}^{A',k'}$ do not overlap, i.e.,

$$\psi_{i,1}^{A,k} > \psi_{i',|\mathbb{S}^{k'}(A')|}^{A',k'} \quad \text{or} \quad \psi_{i,|\mathbb{S}^k(A)|}^{A,k} < \psi_{i',1}^{A',k'}. \tag{4}$$

Let $\Psi = \bigcup_{A\in\mathcal{A}, 1\le k\le|\mathcal{S}(A)|} \Psi^{A,k}$ be the set of all matches of a pattern $\mathbb{S}^k(A)$ in $\mathbb{E}$, for some user activity $A \in \mathcal{A}$.

Assume that $W > 0$ is a given real number and that for each $i = 1, 2, \ldots, |\mathcal{A}|$ and each $k = 1, 2, \ldots, |\mathcal{S}(A^i)|$, there is a nonnegative real number $w(i, k) \ge 0$ such that $\sum_{i=1}^{|\mathcal{A}|} \sum_{k=1}^{|\mathcal{S}(A)|} w(i, k) = W$. For each $\psi_j^{A^i,k} \in \Psi$, we associate a weight $\psi_j^{A^i,k}.weight = w(i, k)$. Let $\Phi \subseteq \Psi$ be a subset of Ψ. The weight of Φ is $w(\Phi) = \sum_{\psi_j^{A^i,k}\in\Phi} w(i, k)$. Φ is said to be compatible if the elements in Φ are mutually compatible.

We study the following optimization problem.

Problem 3 (MaxWCM(Φ, w)): Let $W > 0$ be given, together with weights $w(i, k) \ge 0$ for $i = 1, 2, \ldots, |\mathcal{A}|$ and $k = 1, 2, \ldots, |\mathcal{S}(A^i)|$ such that $\sum_{i=1}^{|\mathcal{A}|} \sum_{k=1}^{|\mathcal{S}(A^i)|} w(i, k) = W$. Let Φ be a subset of Ψ. The weighted maximum compatible match selection problem (MaxWCM) on (Φ, w) seeks for a maximum weight compatible subset $\Phi_{w,opt} \subseteq \Phi$. □

We can apply Algorithm 1 to compute the list $\mathbb{L}^{min,A^i,k}$ of matches in $\Psi^{min,A^i,k}$ for $i = 1, \ldots, |\mathcal{A}|$, $k = 1, \ldots, |\mathcal{S}(A^i)|$. Let $\overline{\Psi} = \bigcup_{A^i\in\mathcal{A}, 1\le k\le|\mathcal{S}(A)|} \{\psi_j^{A^i,k} | \psi_j^{A^i,k} \in \mathbb{L}^{min,A^i,k}\}$. The following theorem shows that an optimal solution of the MaxWCM problem on $(\overline{\Psi}, w)$ is also an optimal solution of the MaxWCM problem on (Ψ, w). Note that $\overline{\Psi}$ is a subset of Ψ whose cardinality is much smaller than that of Ψ.

Theorem 2: The **MaxWCM(Ψ, w)** problem has an optimal solution $\Psi_{w,opt} \subseteq \overline{\Psi}$. Furthermore, such an optimal solution can be obtained by solving the **MaxWCM($\overline{\Psi}, w$)** problem. □

Note that Ψ contains every match of $\mathbb{S}^{A^i,k}$, for every $A^i \in \mathcal{A}$ and every $k \in \{1 \le k \le |\mathcal{S}(A^i)|\}$. Ψ^{min} is a small subset of Ψ, consisting of the minimal matches only. $\overline{\Psi}$ is a small subset of Ψ^{min}, consisting of the representatives of its equivalence classes. Hence $|\overline{\Psi}|$ is much smaller than $|\Psi|$ in general. Theorem 2 shows that we do not lose any important

information by restricting our attention from the very large set Ψ to the much smaller set $\overline{\Psi}$.

4. Inferring Optimal Sequence of User Activity Patterns with a Given Weight (Phase 2A)

In this section, we present an efficient algorithm for solving the MaxWCM problem on $(\overline{\Psi}, w)$, for any given weight w. For ease of presentation, we use a uniform representation of the matches in $\overline{\Psi}$, explained in the following.

We characterize each match $\psi_j^{min,A^i,k} \in \overline{\Psi}$ by a 5-tuple (i,k,α,β,w), where $\alpha = \psi_{j,1}^{min,A^i,k} = \psi_j^{min,A^i,k}[1]$, $\beta = \psi_{j,|\mathbb{S}^k(A^i)|}^{min,A^i,k} = \psi_j^{min,A^i,k}[|\mathbb{S}^k(A^i)|]$, and $\psi_j^{min,A^i,k}.w$ is the weight $w(i,k)$ associated with $\mathbb{S}^k(A^i)$.

Let $m_{i,k} = |\mathbb{L}^{min,A^i,k}|$, $i = 1,\ldots,|\mathcal{A}|$, $k = 1,\ldots,|\mathcal{S}(A^i)|$. Let $M = \sum_{i=1}^{|\mathcal{A}|}\sum_{k=1}^{|\mathcal{S}(A^i)|} m_{i,k}$. Let $\mathbb{M}$ be a 1-D array of M 5-tuples (i,k,α,β,w), each of which corresponds to a match computed. We can sort the array $\mathbb{M}$ according to non-decreasing value of β in $O(M\log M)$ time. Without loss of generality, we assume that array $\mathbb{M}$ is already sorted. We also assume that we have computed a 1-D array p of $M+1$ integers where $p[0] = 0$ and $p[j]$ is the largest integer $t \in \{0,1,\ldots,j-1\}$ such that $\mathbb{M}[t].\beta < \mathbb{M}[j].\alpha$. Here we use the technical convention that $\mathbb{M}[0].\beta = -\infty$. Algorithm 2 presents a solution for the $\mathbf{MaxWCM}(\overline{\Psi}, w)$ problem.

Algorithm 2: OMatch($\mathbb{M}, w$)

Input: $\mathbb{M}$: computed match of $\mathcal{A}$ in $\mathbb{E}$; w: weight function
Output: $\mathbb{M}_w$: max weight sequence of compatible matches

```
OPT[0] ← 0;
for j = 1 to M do
    if (M[j].w + OPT[p[j]] > OPT[j − 1]) then
        OPT[j] ← M[j].w + OPT[p[j]];
    else OPT[j] ← OPT[j − 1];
M_w ← null; j ← M;
while j > 0 do
    if (M[j].w + OPT[p[j]] > OPT[j − 1]) then
        M_w.insert(M[j]); j ← p[j];
    else j ← j − 1;
output M_w.
```

Theorem 3: Algorithm 2 computes a compatible subsequence $\mathbb{M}_w$ of user activity patterns in $\mathbb{M}$ with maximum total weight. Its worst-case running time is $O(M)$. □

Our solution to E2AP and E2A has the following flow. For each $A^i \in \mathcal{A}$, $i = 1,2,\ldots,|\mathcal{A}|$, we know the possible patterns $\mathcal{S}(A^i) = \{\mathbb{S}^1(A^i), \mathbb{S}^2(A^i), \ldots, \mathbb{S}^{|\mathcal{S}(A^i)|}(A^i)\}$. Given the sequence $\mathbb{E}$ of device events, by repeated applications of Algorithm 1 for each $A^i \in \mathcal{A}, i = 1,2,\ldots,|\mathcal{A}|$ and each $k \in \{0,1,2,\ldots,|\mathcal{S}(A^i)|\}$, we can compute $\mathbb{L}^{min,A^i,k}$ of representative matches in $\Psi^{min,A^i,k}$ for each i and k. The union of these lists is $\overline{\Psi}$. Given the weights $w(i,k)$, we can apply Algorithm 2 to compute $\mathbb{M}_w$. From $\mathbb{M}_w$, we obtain a sequence of $|\mathbb{M}_w|$ user activity patterns $\mathbb{S}^w = (\mathbb{S}^w[1], \mathbb{S}^w[2], \ldots, \mathbb{S}^w[|\mathbb{M}_w|])$, where $\mathbb{S}^w[j] = \mathbb{S}^{A^{\mathbb{M}_w[j].i},\mathbb{M}_w[j].k}$, $j = 1,2,\ldots,|\mathbb{M}_w|$. Ignoring the patterns, we obtain a sequence of $|\mathbb{M}_w|$ user activities $\mathbb{A}^w = (\mathbb{A}^w[1], \mathbb{A}^w[2], \ldots, \mathbb{A}^w[|\mathbb{M}_w|])$, where $\mathbb{A}^w[j] = A^{\mathbb{M}_w[j].i}$, $j = 1,2,\ldots,|\mathbb{M}_w|$. We use $\mathbb{S}^w$ and $\mathbb{A}^w$ as our solutions for E2AP and E2A, respectively.

Example 5: Let $\mathbb{A} = (A^1, A^2, A^2)$ be a sequence of user activities, where A^1 and A^2 are as in the example setting. Applying the AkMatch algorithm, we get $\mathbb{L}^{min,A^1,1} = ((1,2),(3,4),(7,8))$, $\mathbb{L}^{min,A^2,1} = ((3,4,6),(7,8,9))$, and $\mathbb{L}^{min,A^2,2} = ((1,2),(3,4),(7,8))$. Putting these 8 matches into array $\mathbb{M}$ and sorting according to the β field, we have $\mathbb{M}[1] = (1,1,1,2,w(1,1))$, $\mathbb{M}[2] = (2,2,1,2,w(2,2))$, $\mathbb{M}[3] = (1,1,3,4,w(1,1))$, $\mathbb{M}[4] = (2,2,3,4,w(2,2))$, $\mathbb{M}[5] = (2,1,3,6,w(2,1))$, $\mathbb{M}[6] = (1,1,7,8,w(1,1))$, $\mathbb{M}[7] = (2,2,7,8,w(2,2))$, $\mathbb{M}[8] = (2,1,7,9,w(2,1))$. We also have $p[0] = p[1] = p[2] = 0$, $p[3] = p[4] = p[5] = 2$, $p[6] = p[7] = p[8] = 5$.

Suppose $w(1,1) = 1.4$, $w(2,1) = 1.6$, $w(2,2) = 0$. The OMatch algorithm will compute $\mathbb{M}_w = (\mathbb{M}[1], \mathbb{M}[5], \mathbb{M}[8])$. Note that $\mathbb{M}[1]$ corresponds to $\mathbb{S}^1(A^1)$, $\mathbb{M}[5]$ corresponds to $\mathbb{S}^1(A^2)$, and $\mathbb{M}[8]$ corresponds to $\mathbb{S}^1(A^2)$. This leads to the sequence of user activity patterns $\mathbb{S}^w = (\mathbb{S}^1(A^1), \mathbb{S}^1(A^2), \mathbb{S}^1(A^2))$ as a solution to E2AP and the sequence of user activities $\mathbb{A}^w = (A^1, A^2, A^2)$ as a solution to E2A. Note that the edit distance between $\mathbb{E}$ and $\mathbb{S}^1(A^1)\|\mathbb{S}^1(A^2)\|\mathbb{S}^1(A^2)$ is 1.

Suppose $w(1,1) = 1.6$, $w(2,1) = 1.4$, $w(2,2) = 0$. The OMatch algorithm will compute $\mathbb{M}_w = (\mathbb{M}[1], \mathbb{M}[3], \mathbb{M}[6])$. This leads to the sequence of user activity patterns $\mathbb{S}^w = (\mathbb{S}^1(A^1), \mathbb{S}^1(A^1), \mathbb{S}^1(A^1))$ as a solution to E2AP and the sequence of user activities $\mathbb{A}^w = (A^1, A^1, A^1)$ as a solution to E2A. Note that the edit distance between $\mathbb{E}$ and $\mathbb{S}^1(A^1)\|\mathbb{S}^1(A^1)\|\mathbb{S}^1(A^1)$ is 3. □

Example 5 shows that the value of the weights has a big impact on the quality of $\mathbb{S}^w$ ($\mathbb{A}^w$, respectively) as a solution to E2AP (E2A, respectively). In Phase 2B, we use unsupervised learning to compute a good weight, by minimizing a properly defined loss function.

5. Learning Optimal Weights (Phase 2B)

In the previous section, we have seen that w uniquely decides $\mathbb{M}_w$, which in turn uniquely decides $\mathbb{S}^w$ and $\mathbb{A}^w$, as our solutions to E2AP and E2A, respectively. In general, $\mathbb{S}^w$ and $\mathbb{S}^{w'}$ ($\mathbb{A}^w$ and $\mathbb{A}^{w'}$, respectively) are not equally good solutions to E2AP (E2A, respectively) when $w \neq w'$, as illustrated in Example 5.

In this section, we present an unsupervised learning approach to learn a proper weight assignment. Both supervised and unsupervised learning can be used. Since the E2AP problem asks for activity patterns rather than activities, we can define a loss function that makes unsupervised learning more adaptive. Hence we focus on unsupervised learning in this paper. Due to space limitations, we leave the application of supervised learning as well as distributed unsupervised learning to this problem as future research directions.

In Section 5-A, we define the loss function as the edit distance between $\mathbb{E}$ and the concatenation of the activity patterns computed. The loss function is a non-differentiable function of the continuous variables corresponding to the weights. In Section 5-B, we design an exact algorithm to

minimize the loss function with all but one variable fixed. This algorithm is used as a subroutine in Section 5-C to design a coordinated descent algorithm to minimize the loss function.

A. Loss Function and Optimization Problem Formulation

Let $\mathbb{E}^w$ be the sequence of device events obtained by concatenating the activity patterns in $\mathbb{S}^w$:

$$\mathbb{E}^w = \mathbb{S}^w[1] \| \mathbb{S}^w[2] \| \cdots \| \mathbb{S}^w[|\mathbb{M}_w|] \ . \tag{5}$$

Without any knowledge of the ground truth, we define the loss function for parameter w as the edit distance between $\mathbb{E}^w$ and $\mathbb{E}$, denoted by $d(\mathbb{E}^w, \mathbb{E})$. Here we assume that the operations `deletion`, `insertion`, and `substitution` all have costs equal to 1.

Let $n = |\mathcal{A}|$; $n_j = |\mathcal{S}(A^j)|$ for $j = 1, 2, \ldots, n$; and $N = \sum_{j=1}^{n} n_j$. Then w consists of N variables. A natural approach to obtaining the optimal values of w is to solve the following optimization problem.

$$\underset{w}{\text{minimize}}\, f(w) = d(\mathbb{E}^w, \mathbb{E}) \tag{6}$$

$$\text{subject to} \sum_{i=1}^{n} \sum_{k=1}^{|\mathcal{S}(A^i)|} w(i,k) = N, \tag{6a}$$

$$w(i,k) \geq 0, \forall i = 1, \ldots, n, \forall k = 1, \ldots, |\mathcal{S}(A^i)|. \tag{6b}$$

Problem (6) is difficult to solve, as the objective function is non-differentiable and non-convex [3]. In an effort to design a coordinated descent algorithm for solving (6), we study an optimization problem where we minimize the objective function over one of the N variables. We formally define this problem in the following, where $w(\underline{i}, \underline{k})$ is the variable, for a chosen pair $(\underline{i}, \underline{k})$:

$$\underset{w(\underline{i},\underline{k})}{\text{minimize}}\, f(w) = d(\mathbb{E}^w, \mathbb{E}) \tag{7}$$

$$\text{subject to}\, w(\underline{i}, \underline{k}) \geq 0. \tag{7a}$$

We call (7) the 1-D problem, and (6) the N-D problem. We present an exact algorithm for solving (7) and a coordinate descent algorithm for solving (6).

B. Exact Algorithm for 1-D Optimization

We design an algorithm named SLN-1D that computes an optimal solution of the 1-D problem, and list it as Algorithm 3.

Algorithm 3 performs many rounds of Algorithm 2, by tracking the trend of the computed matches with the value of $w(\underline{i}, \underline{k})$ taking on larger and larger values. Here $\mathbb{M}$ and p are the same as in Algorithm 2.

The key observation behind the design of Algorithm 3 is that the objective function $f(w)$ in (7) is a *step function* of $w(\underline{i}, \underline{k})$ with a finite number of different function values, each of which is defined on an interval for $w(\underline{i}, \underline{k})$. We can evaluate $f(w)$ in an interior point and on the boundaries of each of these intervals. We find the boundaries of these intervals by performing a left-to-right scan.

We start the scan at $w(\underline{i}, \underline{k}) = 0$ (Line 1). For each value x of $w(\underline{i}, \underline{k})$, we compute real numbers μ and ν where $\mu > x$ and $\nu = \frac{x+\mu}{2}$ such that

(i) $\mathbb{M}^w$ does not change when $w(\underline{i}, \underline{k})$ is varying in the interval (x, μ).

(ii) $\mathbb{M}^w$ will change when $w(\underline{i}, \underline{k})$ is increased from x to μ or from x to μ plus an infinitesimal amount.

Algorithm 3: SLN-1D($\mathbb{M}, w, \underline{i}, \underline{k}$)

Input: $\mathbb{M}$: computed match of $\mathcal{A}$ in $\mathbb{E}$; w: weight vector; $\underline{i}$, $\underline{k}$: index pair for variable $w(\underline{i}, \underline{k})$

Output: x_{opt}: optimal value for $w(\underline{i}, \underline{k})$; f_{opt}: loss function value with $w(\underline{i}, \underline{k})$ at optimal value

```
1  x ← 0; x_opt ← 0; w(i,k) ← x; f_opt ← f(w); E_old ← E^w;
2  OPT[0] ← 0; a[0] ← 0; σ ← ∞;
3  for j := 1 to M do
4      if (M[j].A = A^i and M[j].k = k) then
5          δ[j] ← 1;            /* M[j].w is variable */
6      else
7          δ[j] ← 0;            /* M[j].w is constant */
8      if (M[j].w + OPT[p[j]] > OPT[j − 1]) then
9          OPT[j] ← M[j].w + OPT[p[j]];
10         a[j] ← a[p[j]] + δ[j];
11         if a[p[j]] + δ[j] < a[j − 1] then
12             σ ← min{σ, (M[j].w+OPT[p[j]]−OPT[j−1]) / (a[j−1]−a[p[j]]−δ[j])};
13     else if (M[j].w+OPT[p[j]]<OPT[j−1]) then
14         OPT[j] ← OPT[j − 1];
15         a[j] ← a[j − 1];
16         if a[p[j]] + δ[j] > a[j − 1] then
17             σ ← min{σ, (M[j].w+OPT[p[j]]−OPT[j−1]) / (a[j−1]−a[p[j]]−δ[j])};
18     else
19         if a[p[j]] + δ[j] ≤ a[j − 1] then
20             OPT[j] ← OPT[j − 1];
21             a[j] ← a[j − 1];
22         else
23             OPT[j] ← M[j].w + OPT[p[j]];
24             a[j] ← a[p[j]] + δ[j];
25 if σ = ∞ then { σ ← 2; done ← true; }
26 μ ← x + σ; ν ← x + σ/2;
27 w(i,k) ← ν; E_new ← E^w;
28 if (E_new ≠ E_old) or (ν = σ/2) then
29     f_new ← f(w); E_old ← E_new;
30     if f_new < f_opt or (f_new = f_opt and ν = σ/2) then
31         x_opt ← ν; f_opt ← f_new;
32 w(i,k) ← μ; E_new ← E^w;
33 if (E_new ≠ E_old) then
34     f_new ← f(w); E_old ← E_new;
35     if f_new < f_opt then { x_opt ← μ; f_opt ← f_new; }
36 if (not done) then { x ← μ; goto 2; }
37 output x_opt, f_opt.
```

We evaluate $f(w)$ by setting $w(\underline{i}, \underline{k})$ to ν and μ, respectively. Then repeat the process with x set to μ, as long as such a μ exists. When such a μ does not exist, we set $\mu \leftarrow x + 2$ and $\nu \leftarrow x + 1$, and evaluate $f(w)$ by setting $w(\underline{i}, \underline{k})$ to ν and μ, respectively. Lines 2-25 carry out the computation of $\sigma = \mu - x$ for the current value of x. In Line 26, we set $\mu \leftarrow x + \sigma$ and $\nu \leftarrow x + \sigma/2$. Lines 27-35 perform the function evaluations at ν and μ, and related book-keeping operations.

The computation of $\sigma = \mu - x$ from x is carried out by operations similar to the operations in the dynamic programming algorithm for weighted interval scheduling, *i.e.*, Lines 1-6 of Algorithm 2. There are additional operations needed to compute σ.

Before proceeding with the explanation of the algorithm, we describe the meaning of two variables: array $\delta[]$ and array $a[]$. For $j = 1, 2, \ldots, M$, $\delta[j] = 1$ if $\mathbb{M}[j].w$ is the variable

weight $w(\underline{i},\underline{k})$, $\delta[j]=0$ otherwise. For $j=1,2,\ldots,M$, we use $a[j]$ to denote the number of selected matches/intervals in the optimal solution of $\mathbb{M}(1:j)$ whose weight is $w(\underline{i},\underline{k})$, when $w(\underline{i},\underline{k})$ is set to x plus an infinitesimal amount.

In Line 2, we set $OPT[0]\leftarrow 0$, $a[0]\leftarrow 0$, $\sigma\leftarrow\infty$. The **for** loop in Lines 3-24 computes the values for $\delta[j]$, $OPT[j]$, $a[j]$, and reduces σ accordingly. The body of the **for** loop starts with the computation of $\delta[j]$ in Lines 4-7. This is followed by three mutual exclusive parts: Lines 9-12 (when $\mathbb{M}[j].w+OPT[p[j]]>OPT[j-1]$), Lines 14-17 (when $\mathbb{M}[j].w+OPT[p[j]]<OPT[j-1]$), and Lines 19-24 (when $\mathbb{M}[j].w+OPT[p[j]]=OPT[j-1]$).

When $\mathbb{M}[j].w+OPT[p[j]]>OPT[j-1]$, the optimal solution of $\mathbb{M}(1:j)$ consists of $\mathbb{M}[j]$ and the optimal solution of $\mathbb{M}(1:p[j])$. This will not change even when $w(\underline{i},\underline{k})$ is increased by an infinitesimal amount, due to the strict inequality. Hence $a[j]$ is computed according to Line 10. If $a[p[j]]+\delta[j]<a[j-1]$, the inequality $\mathbb{M}[j].w+OPT[p[j]]>OPT[j-1]$ will no longer be true when $w(\underline{i},\underline{k})$ is increased by an amount equal to $\frac{\mathbb{M}[j].w+OPT[p[j]]-OPT[j-1]}{a[j-1]-a[p[j]]-\delta[j]}$. Therefore we set σ to the smaller of its current value and $\frac{\mathbb{M}[j].w+OPT[p[j]]-OPT[j-1]}{a[j-1]-a[p[j]]-\delta[j]}$ in Line 12.

When $\mathbb{M}[j].w+OPT[p[j]]<OPT[j-1]$, the optimal solution of $\mathbb{M}(1:j)$ is the optimal solution of $\mathbb{M}(1:j-1)$. This will not change even when $w(\underline{i},\underline{k})$ is increased by an infinitesimal amount, due to the strict inequality. Hence $a[j]$ is computed according to Line 15. If $a[p[j]]+\delta[j]>a[j-1]$, we will have $\mathbb{M}[j].w+OPT[p[j]]>OPT[j-1]$ when $w(\underline{i},\underline{k})$ is increased by an amount equal to an infinitesimal plus $\frac{\mathbb{M}[j].w+OPT[p[j]]-OPT[j-1]}{a[j-1]-a[p[j]]-\delta[j]}$. Therefore we set σ to the smaller of its current value and $\frac{\mathbb{M}[j].w+OPT[p[j]]-OPT[j-1]}{a[j-1]-a[p[j]]-\delta[j]}$ in Line 17.

When $\mathbb{M}[j].w+OPT[p[j]]=OPT[j-1]$, the optimal solution of $\mathbb{M}(1:j)$ is the optimal solution of $\mathbb{M}(1:j-1)$, with an objective function value equal to $OPT[j-1]=\mathbb{M}[j].w+OPT[p[j]]$. If $a[p[j]]+\delta[j]\le a[j-1]$, we will have $\mathbb{M}[j].w+OPT[p[j]]\le OPT[j-1]$ when $w(\underline{i},\underline{k})$ is increased by an infinitesimal amount. Note that $\mathbb{M}[j].w+OPT[p[j]]\le OPT[j-1]$ implies that the optimal solution for $\mathbb{M}(1:j)$ is the optimal solution for $\mathbb{M}(1:j-1)$. Therefore we choose the assignment of $OPT[j]$ in Line 20 and set the value of $a[j]$ according to Line 21. If $a[p[j]]+\delta[j]>a[j-1]$, we will have $\mathbb{M}[j].w+OPT[p[j]]>OPT[j-1]$ when $w(\underline{i},\underline{k})$ is increased by an infinitesimal amount. Note that $\mathbb{M}[j].w+OPT[p[j]]>OPT[j-1]$ implies that the optimal solution for $\mathbb{M}(1:j)$ consists of $\mathbb{M}[j]$ and the optimal solution for $\mathbb{M}(1:p[j])$. Therefore we choose the assignment of $OPT[j]$ in Line 23 and set the value of $a[j]$ according to Line 24.

If we find $\sigma=\infty$ in Line 25, we know that x is the left end of the rightmost interval for $w(\underline{i},\underline{k})$. However, we do not know whether the function is right-continuous at x. Therefore, we set σ to 2 (which can be replaced by any other positive number). In Line 26, we set $\mu\leftarrow x+\sigma$ and $\nu\leftarrow x+0.5\sigma$. Lines 27-35 are straightforward which do not need explanation. If the current x is not the left end of the rightmost interval, we set $x\leftarrow\mu$ and repeat the process.

Theorem 4: For any given value of w and index pair $(\underline{i},\underline{k})$, Algorithm 3 terminates with an optimal solution to (7). The worst-case time complexity of Algorithm 3 is $O(M^2)$. □

C. Coordinate Descent for N-D Optimization

The optimization problem defined by (6) is difficult to solve for two reasons: (i) The objective function is non-differentiable; (ii) The optimization problem is non-convex. Therefore we tackle the problem via a round-robin coordinate descent approach. This is presented in Algorithm 4.

Algorithm 4: SLN-ND($\mathbb{M}$)

Input: $\mathbb{M}$: representative minimal matches of $\mathcal{A}$ in $\mathbb{E}$.
Output: w: optimal weight vector; $\mathbb{S}^w$: corresponding solution to E2AP.

/* initialization */
1 **for** $i:=1$ **to** n **do** {**for** $k:=1$ **to** n_i **do** $w[i,k]\leftarrow 1$;}
2 $done\leftarrow$ **false**; $f_{opt}\leftarrow f(w)$;
/* coordinate descent */
3 **while** ($done=$ **false**) **do**
  /* normalize the weights */
4 $W\leftarrow 0$;
5 **for** $i:=1$ **to** n **do**
6 **for** $k:=1$ **to** n_i **do** $W\leftarrow W+w[i,k]$;
7 **for** $i:=1$ **to** n **do**
8 **for** $k:=1$ **to** n_i **do** $w[i,k]\leftarrow\frac{N\times w[i,k]}{W}$;
9 $done\leftarrow$ **true**;
10 **for** $i:=1$ **to** n **do**
  /* 1D minimization */
11 **for** $k:=1$ **to** n_i **do**
12 $x_{old}\leftarrow w(i,k)$;
13 $x_{new}\leftarrow\arg\min_{w(i,k)\ge 0} f(w)$; $f_{new}\leftarrow f(w)$;
14 **if** $f_{new}<f_{opt}$ **then**
15 $done\leftarrow$ **false**; $w(i,k)\leftarrow x_{new}$; $f_{opt}\leftarrow f_{new}$;
16 **else** $w(i,k)\leftarrow x_{old}$;
17 **output** w, $\mathbb{S}^w$.

The following explains the main steps of Algorithm 4. Line 1 performs the initialization of the weights. It then performs coordinate descent until convergence. Lines 4-8 perform normalization so that the N non-negative weights sum up to N. Instead of normalizing after each one-dimensional minimization, we normalize it after one round-robin over all N variables. Lines 10-16 perform one round of coordinated descent. For each (i,k), we perform 1-D minimization of $f(w)$ over $w(i,k)$ by calling Algorithm 3.

Example 6: Assume the same setting as in Example 5. Let $\mathbb{M}$ contain the 8 matches computed. We apply Algorithm 4 to this setting, with the initial weights $w(1,1)=1$, $w(2,1)=1$, $w(2,2)=1$. The objective function value is $f(w)=3$, with $\mathbb{M}_w=(\mathbb{M}[1],\mathbb{M}[3],\mathbb{M}[6])$.

Algorithm 4 performs minimization over $w(1,1)$, there is no improvement. It then performs minimization over $w(2,1)$. The objective function value is reduced to 1 with $w(1,1)=1$, $w(2,1)=2$, $w(2,2)=1$. It then performs minimization over $w(2,2)$, there is no improvement. After normalization, we have $w(1,1)=\frac{3}{4}$, $w(2,1)=\frac{3}{2}$, $w(2,2)=\frac{3}{4}$. The algorithm performs minimization over $w(1,1)$, $w(2,1)$, and $w(2,2)$. None of these produce any improvement. The algorithm terminates, with $w(1,1)=\frac{3}{4}$, $w(2,1)=\frac{3}{2}$, $w(2,2)=\frac{3}{4}$; $f(w)=1$; $\mathbb{M}_w=(\mathbb{M}[1],\mathbb{M}[5],\mathbb{M}[8])$; and $\mathbb{S}^w=(\mathbb{S}^1(A^1),\mathbb{S}^1(A^2),\mathbb{S}^1(A^2))$. □

Theorem 5: Algorithm 4 stops after a finite number of iterations. Let w be the weight at the end of the algorithm. Then $f(w)$ cannot be reduced by minimizing over any single variable $w(i,k)$. □

Algorithm 4 does not guarantee finding an optimal solution to (6). Similar to most machine learning algorithms [14], Algorithm 4 produces a solution to the (6) that cannot be improved by optimizing along any of the coordinates. We do not know a theoretical bound on the performance gap of the algorithm. However, our extensive experiments (presented in Section 6) show that the algorithm performs well.

Besides proving that the algorithm converges in a finite number of iterations, we do not have a theoretical bound on the worst-case running time of the algorithm. Given the non-convex nature of the problem, it is unlikely to compute an optimal solution in polynomial time.

6. Performance Evaluations

We implemented our scheme and compared it with IoTMosaic [18]. We did not compare our proposed solution with [12] or Peek-a-Boo [1], since many devices used in [12] are simple sensors without Internet connectivity and Peek-a-Boo [1] infers user activities from the states of devices and sensors, rather than solely from the sequence of device events. We use `E2AP` to denote our scheme, and `IoTMosaic` to denote the scheme in [18]. We studied the performance on both real data from our smart home testbed, and synthetic data generated following the patterns observed in the real data.

In Section 6-A, we present the evaluation setup and metrics. In Section 6-B, we present our evaluation results and our observations/analyses. These results show that `E2AP` exhibits high accuracy and stability, and outperforms `IoTMosaic`.

A. Evaluation Setup and Metrics

We used the data collected from a smart home, involving 21 distinct user activities and 25 distinct device events from 11 different IoT devices. We refer the readers to [18] for detailed descriptions of the experiment setup.

Experimental data were collected over a period of two months, with a total of $2,959$ user activities, and 15,420 corresponding device events (computed from network traffic using IoTAthena [17]). We call these data real data and denote them as `real`. We study three cases: (i) 387 user activities ($2,013$ events) in the first week, (ii) $1,494$ user activities ($7,934$ events) in the first month, (iii) $2,959$ user activities ($15,420$ events) throughout the experiment. In the `real` data, two device events from two IoT devices (Ring doorbell and Ring spotlight) could be delayed, due to devices on sleep mode. No device malfunctioned during the experiment.

We also wrote a test case generator, based on the characteristics of the `real` data collected. In addition to delayed device events, we added scenarios where up to two devices malfunctioned during part of the time. We call these data synthetic data, and denote them by `synth` where no devices malfunctioned, and by `synth-MF` where up to two devices malfunctioned during part of the time.

We ran `E2AP` and `IoTMosaic` on both the real data and the synthetic data. Let $\mathbb{E}$ denote the input sequence of device events, $\mathbb{A}$ denote the input sequence of user activities, w denote the weight vector computed, $\mathbb{S}^w$ denote the sequence of activity patterns computed, $\mathbb{E}^w$ denote the sequence of device events by the concatenation of the elements of $\mathbb{S}^w$, and $\mathbb{A}^w$ denote the sequence of user activities computed. For `IoTMosaic`, we measure the number of undetected user activities (denoted by FN), the number of falsely detected user activities (denoted by FP), and the accuracy. For `E2AP`, we measure the edit distance between $\mathbb{E}^w$ and $\mathbb{E}$ (denoted by $d(E)$), the edit distance between $\mathbb{A}^w$ and $\mathbb{A}$ (denoted by $d(A)$), the number of undetected user activities (denoted by FN), the number of falsely detected user activities (denoted by FP), and the accuracy. Since $d(E)$ and $d(A)$ tend to increase as the lengths of the sequences increase, we introduce $d_N(E) = d(E)/|\mathbb{E}|$ and $d_N(A) = d(A)/|\mathbb{A}|$ as a form of normalization.

B. Evaluation Results and Observations

Table 3 shows the evaluation results. For each of the three sizes for `real` data (first week, first month, two months), the corresponding entries in the table are the results of a single run of the denoted algorithm. For each of the five sizes for `synth` and `synth-MF`, we generated 100 test cases. These 100 test cases have the *same number* of user activities, but have *different sequences* of user activities (as these are randomly generated), hence having a different number of device events (as the device events are triggered by different user activities). Each entry for #Event shows the minimum and maximum numbers of events (over 100 test cases). All other entries are the average over 100 test cases. We observe that `E2AP` has a higher accuracy than `IoTMosaic` and exhibits more stability.

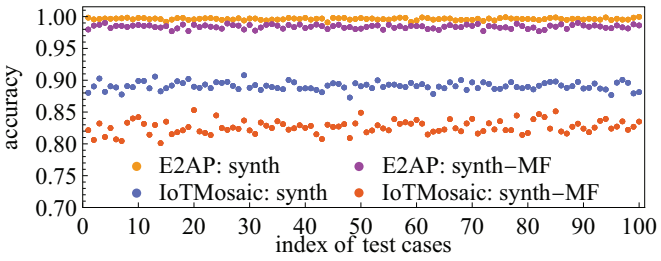


Fig. 1. Performances of `E2AP` and `IoTMosaic` on `synth/synth-MF` test cases where each test case consists of a distinct sequence of 2959 user activities and a corresponding sequence of device events. Each dot in the figure shows the accuracy achieved by the indicated algorithm on a distinct test case.

Fig. 1 illustrates the accuracy of `IoTMosaic` and `E2AP` on 200 synthetically generated test cases (100 test cases with device malfunctions, and 100 test cases without device malfunctions) where each test case consists of a distinct sequence of $2,959$ user activities and a corresponding sequence of device events, following the distribution of our real data. The lengths of the sequences of device events vary between $13,382$ and $14,264$, as shown in Table 3.

We observe that both `IoTMosaic` and `E2AP` are quite accurate. However, `E2AP` is consistently more accurate than `IoTMosaic`. For the 100 test cases without device malfunctions, the accuracies of `E2AP` fall in the interval

TABLE 3
RESULTS ON REAL DATA (ONE CASE PER ROW) AND SYNTHETIC DATA (AVERAGE OVER 100 CASES PER ROW)

Data Type	#Act	#Event	IoTMosaic ($k \leq 5$)			E2AP						
			FN	FP	Accuracy	FN	FP	Accuracy	$d(A)$	$d_N(A)$	$d(E)$	$d_N(E)$
real (week)	387	2013	6	2	97.92%	2	2	98.97%	4	0.0103	22	0.0109
real (month)	1494	7934	32	13	96.98%	2	12	99.06%	14	0.0094	104	0.0131
real (full)	2959	15420	42	18	97.97%	2	16	99.39%	18	0.0061	142	0.0092
synth	387	[1666, 1941]	38.76	0.44	88.77%	1.50	0.82	99.61%	1.52	0.0039	2.62	0.0014
synth	1494	[6708, 7202]	147.50	0.92	88.99%	5.23	2.58	99.60%	5.96	0.0040	10.58	0.0015
synth	2959	[13382, 14264]	290.08	1.36	89.09%	9.27	4.58	99.60%	11.05	0.0037	20.33	0.0015
synth	10000	[46122, 47496]	984.61	4.67	89.04%	29.77	12.41	99.60%	40.46	0.0040	73.21	0.0016
synth	30000	[139427, 141632]	2936.63	12.31	89.11%	82.91	33.19	99.61%	117.04	0.0039	212.84	0.0015
synth-MF	387	[1337, 1590]	49.45	19.49	81.69%	3.90	3.38	98.33%	6.47	0.0167	1.99	0.0013
synth-MF	1494	[5645, 6009]	183.28	68.78	82.60%	10.89	8.44	98.32%	25.02	0.0167	8.50	0.0015
synth-MF	2959	[11161, 11855]	361.27	137.15	82.64%	19.10	14.62	98.05%	47.07	0.0159	15.67	0.0014
synth-MF	10000	[38312, 39401]	1221.87	460.44	82.65%	54.78	39.67	98.44%	155.74	0.0156	53.31	0.0014
synth-MF	30000	[115946, 117785]	3648.52	1385.25	82.71%	156.95	111.75	98.45%	465.13	0.0155	160.43	0.0014

$[99.09\%, 99.97\%]$ with a mean of 99.63% and a standard deviation of 0.0016, compared to that of `IoTMosaic` in the interval $[87.28\%, 90.78\%]$ with a mean of 89.09% and a standard deviation of 0.0063. For the 100 test cases with device malfunctions, the accuracies of `E2AP` fall in the interval $[97.70\%, 99.02\%]$ with a mean of 98.41% and a standard deviation of 0.0030, compared to that of `IoTMosaic` in the interval $[80.11\%, 85.31\%]$ with a mean of 82.65% and a standard deviation of 0.0103.

The advantage of `E2AP` over `IoTMosaic` is more significant over the combined 200 test cases. The accuracies of `E2AP` fall in the interval $[97.70\%, 99.97\%]$ with a mean of 99.02% and a standard deviation of 0.0066, compared to that of `IoTMosaic` in the interval $[80.11\%, 90.78\%]$ with a mean of 85.87% and a standard deviation of 0.0333. We observe that `E2AP` is 15% more accurate than `IoTMosaic` and 5 times more stable than `IoTMosaic`.

As we discussed earlier, `IoTMosaic` gives higher priorities to exact matches of a signature over partial matches of a signature. When a device malfunctions, events corresponding to this malfunctioned device will not appear in the observed sequence of device events. Therefore, in the presence of malfunctioning devices, a user activity may only trigger a subsequence of its full signature. Therefore the number of exact matches will be reduced. This is the root cause for the lower accuracy and higher instability of `IoTMosaic`. In contrast, `E2AP` can intelligently adapt to such situations by learning the proper weights for all possible patterns.

We conducted more experiments with different synthetic datasets, and found that the results are consistent with those presented in Table 3 and Fig. 1. In a nutshell, `E2AP` is consistently accurate and stable across different experiment scenarios and varying test cases. On the other hand, the accuracy of `IoTMosaic` decreases when the number of missing events increases, which conforms to our analysis above.

7. CONCLUSIONS

In this paper, we studied the problem of inferring a sequence of user activities together with their patterns from a sequence of device events in a smart home setting. We designed a novel two-phase scheme for solving this problem. A key contribution of this paper is the unsupervised learning algorithm which helps make the inference more adaptive to varying scenarios. No existing algorithm can be directly applied to minimize the non-differentiable and non-convex loss function for our problem. We designed a novel algorithm for minimizing the loss function over one variable, which is a central component in our unsupervised learning algorithm. Extensive evaluations show that our algorithm is significantly more robust and accurate than the state-of-the-art algorithm. Future work includes evaluating the limitations of our scheme in more smart homes as well as exploring its applications in home security. Another line of research is to explore other machine learning techniques to solve this problem.

REFERENCES

[1] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-Boo: I see your smart home ativities, even encrypted!" in *Proc. of ACM WiSec*, 2020.

[2] V. Bhosale, L.-D. Carli, and I. Ray, "Detection of anomalous user activity for home IoT devices," in *Proc. of IoTBDS*, 2021.

[3] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[4] H. Chi, Q. Zeng, X. Du, and L. Luo, "PFirewall: Semantics-aware customizable data flow control for home automation systems," in *Proc. of NDSS*, 2021.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2009.

[6] S. Feng, P. Setoodeh, and S. Haykin. "Smart home: Cognitive interactive people-centric internet of things," *IEEE Communications Magazine*, 55(2):34 – 39, 2017.

[7] C. Fu, Q. Zeng, and X. Du, "HAWatcher: Semantics-aware anomaly detection for appified smart homes," in *Proc. of USENIX Security*, 2021.

[8] J. Kleinberg and E. Tardos, *Algorithm Design*. Pearson Education, 2006.

[9] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: An analysis of IoT devices on home networks," in *Proc. of USENIX Security*, 2019.

[10] F. S. Lesani, F. F. Ghazvini, and H. Amirkhani, "Smart home resident identification based on behavioral patterns using ambient sensors," *Springer Personal and Ubiquitous Computing*, 25:151–162, 2019.

[11] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi, "HomeSnitch: Behavior

transparency and control for smart home IoT devicess," in *Proc. of ACM WiSec*, 2019.

[12] P. Rashidi, D. J. Cook, L. B. Holder, and M. Schmitter-Edgecombe, "Discovering activities to recognize and track in a smart environment," *IEEE Transactions on Knowledge and Data Engineering*, 23(4):527–539, 2011.

[13] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer IoT devices: A multidimensional, network-informed measurement approach," in *ACM IMC*, 2019.

[14] S. Sun, Z. Cao, H. Zhu, and J. Zhao. "A survey of optimization methods from a machine learning perspective," *IEEE Trans. on Cybernetics*, 50(8):3668–3681, 2020.

[15] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "PingPong: Packet-level signatures for smart home device events," in *Proc. of NDSS*, 2019.

[16] B. van der Waerden, *Modern Algebra, 2nd ed.* 1949.

[17] Y. Wan, K. Xu, F. Wang, G. Xue, "IoTAthena: Unveiling IoT device activities from network traffic," *IEEE Trans. on Wireless Communications*, 21(1): 651-664, 2021.

[18] Y. Wan, K. Xu, F. Wang, and G. Xue, "IoTMosaic: Inferring user activities from IoT network traffic in smart homes," in *Proc. of IEEE INFOCOM*, 2022.

[19] R. Want, B. Schilit, and S. Jenson, "Enabling the Internet of Things," *Computer*, 48(1):28 – 35, 2015.

[20] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "HoMonit: Monitoring smart home apps from encrypted traffic," in *Proc. of ACM CCS*, 2018.

APPENDIX

Proof of Lemma 1: By Definition 3, two matches of $\mathbb{S}^k(A)$ in $\mathbb{E}$ are equivalent if and only if their intervals are the same. Each interval must be in the form $[\alpha, \beta]$ where α and β are integers such that $1 \leq \alpha \leq \beta \leq m$. The number of such intervals is $m(m+1)/2$. This proves the upper-bound on the number of equivalence classes of $\Psi^{A,k}$.

Let $\psi_i^{min,A,k}$ and $\psi_{i'}^{min,A,k}$ be two non-equivalent elements of $\Psi^{min,A,k}$. Let $[\alpha, \beta]$ and $[\alpha', \beta']$ be the intervals of $\psi_i^{min,A,k}$ and $\psi_{i'}^{min,A,k}$, respectively. We claim that $\alpha \neq \alpha'$. Suppose to the contrary that $\alpha = \alpha'$. If $\beta = \beta'$, we conclude that $\psi_i^{min,A,k} \equiv \psi_{i'}^{min,A,k}$; If $\beta < \beta'$, we conclude that $\psi_{i'}^{min,A,k}$ is not minimal; If $\beta > \beta'$, we conclude that $\psi_i^{min,A,k}$ is not minimal. Therefore we have proved that $\alpha \neq \alpha'$. This fact implies that the number of equivalence classes of $\Psi^{min,A,k}$ is upper-bounded by m. □

Proof of Theorem 1: Algorithm 1 is a modification of the algorithm for computing an LCS of two sequences $\mathbb{E}(start : m)$ and $\mathbb{S}^k(A)$. The difference lies in (i) we are only interested in computing an LCS that is identical to $\mathbb{S}^k(A)$, not any proper subsequence of $\mathbb{S}^k(A)$; and (ii) we compute multiple matches, rather than one. Therefore we omit the details that are well-known in the correctness of the algorithm for LCS [5].

We use η to denote $|\mathbb{S}^k(A)|$. Computing one LCS requires $O(m\eta)$ time in the worst-case. The algorithm computes $O(m)$ LCSs. Hence the worst-case running time is $O(m^2\eta)$.

If $\text{LCS}(\mathbb{E}, \mathbb{S}^k(A)) \neq \mathbb{S}^k(A)$, Algorithm 1 outputs a **null** list, as $\Psi^{min,A,k} = \Psi^{A,k} = \emptyset$ in this case. In the rest of the proof, we assume $\text{LCS}(\mathbb{E}, \mathbb{S}^k(A)) = \mathbb{S}^k(A)$. Hence the condition in Line 11 is true at least once during the execution.

Each time when the condition in Line 11 is true, we know that $\mathbb{E}(start : i-1)$ contains no subsequence that is identical to $\mathbb{S}^k(A)$, and that $\mathbb{E}(start : i)$ contains a subsequence that is identical to $\mathbb{S}^k(A)$. Therefore Lines 12-19 correctly compute a match $\psi_l = (\psi_l[1], \psi_l[2], \ldots, \psi_l[\eta])$ of $\mathbb{S}^k(A)$ in $\mathbb{E}(start : i)$.

Let $x15$ and $x18$ denote the number of times Line 15 and Line 18 are executed (with $start$ and l fixed), respectively. Since control enters the **while** loop in Line 13 with col initialized to η, and exits the **while** loop when col is reduced to 0, we have $x15 + x18 = \eta$. Because a match of $\mathbb{S}^k(A)$ in $\mathbb{E}$ is found that lies entirely in $\mathbb{E}(start : i)$, we have $x15 = \eta$. Hence $x18 = 0$. In other words, between the executions of Line 12 and Line 19, Line 15 is executed exactly η times, and Line 18 is executed zero times.

When we trace out a newly computed match of $\mathbb{S}^k(A)$ in $\mathbb{E}$, we first initialize row to i and col to η. If $e_{row} = e_{col}^{A,k}$, we decrement both row and col by 1. Otherwise, we decrement row by 1 and keep col unchanged. Therefore, the match we trace out is the lightest among all matches of $\mathbb{S}^k(A)$ in $\mathbb{E}$ that lie entirely in $\mathbb{E}(start, i)$.

When the condition in Line 11 becomes true for the first time, we know that there is no match of $\mathbb{S}^k(A)$ in $\mathbb{E}$ that lies within $\mathbb{E}(start : i) = \mathbb{E}(1 : i)$. Therefore $\psi_1 = (\psi_1[1], \psi_1[2], \ldots, \psi_1[\eta])$ is the lightest match of $\mathbb{S}^k(A)$ in $\mathbb{E}$, which implies that it is minimal. Let $\phi = (\phi[1], \phi[2], \ldots, \phi[\eta])$ be any minimal match of $\mathbb{S}^k(A)$ in $\mathbb{E}$ that is not equivalent to ψ_1, we must have $\phi[1] \geq \psi_1[1] + 1$. Therefore ψ_2 computed by our algorithm is the lightest minimal match of $\mathbb{S}^k(A)$ in $\mathbb{E}$ that is not equivalent to ψ_1. Similarly, ψ_3 computed by our algorithm is the lightest minimal match of $\mathbb{S}^k(A)$ in $\mathbb{E}$ that is not equivalent to ψ_1 or ψ_2. In general, ψ_l computed by our algorithm is the lightest minimal match of $\mathbb{S}^k(A)$ in $\mathbb{E}$ that is not equivalent to ψ_j for $j = 1, 2, \ldots, l-1$. This completes the proof of the theorem. □

Proof of Theorem 2: Let $\Psi^{min} = \bigcup_{A^i \in \mathcal{A}, 1 \leq k \leq |\mathcal{S}(A^i)|} \Psi^{min,A^i,k}$. We first prove that there is an optimal solution $\Psi_{w,opt}$ of the **MaxWCM**(Ψ, w) problem such that $\Psi_{w,opt} \subseteq \Psi^{min}$.

Let $\Psi_{w,opt}$ be an arbitrary optimal solution for the **MaxWCM**(Ψ, w) problem. If $\Psi_{w,opt} \subseteq \Psi^{min}$, there is nothing to be proved. Let $\psi_j^{A^i,k}$ be a match in $\Psi_{w,opt}$ such that $\psi_j^{A^i,k} \in \Psi^{A^i,k} \setminus \Psi^{min,A^i,k}$. Let $\psi_{j'}^{A^i,k}$ be a minimal match of $\mathbb{S}(A^i)$ in $\mathbb{E}$ such that $[\psi_{j'}^{A^i,k}[1], \psi_{j'}^{A^i,k}[|\mathbb{S}^k(A^i)|]$ is a proper sub-interval of $[\psi_j^{A^i,k}[1], \psi_j^{A^i,k}[|\mathbb{S}^k(A^i)|]$. Since the two matches have the same weight $w(i,k)$, we can obtain another optimal solution by replacing $\psi_j^{A^i,k}$ with $\psi_{j'}^{A^i,k}$. Therefore, we can obtain another optimal solution to the **MaxWCM**(Ψ, w) problem with one fewer non-minimal match than $\Psi_{w,opt}$. Repeating this process for each non-minimal match in $\Psi_{w,opt}$, we will obtain an optimal solution to the **MaxWCM**(Ψ, w) problem consisting of matches in Ψ^{min} only.

WLOG, we will assume that $\Psi_{w,opt} \subseteq \Psi^{min}$ in the rest of this proof. Next, we show that we can transform the optimal solution $\Psi_{w,opt}$ to another optimal solution $\Psi_{w,opt} \subseteq \overline{\Psi}$.

Suppose there is an element $\psi_j^{min,A^i,k} \in \Psi_{w,opt}$ that is not in $\overline{\Psi}$. Let $\psi_{j_{lightest}}^{min,A^i,k}$ be the lightest element in the equivalence class which contains $\psi_j^{min,A^i,k}$. Then we can replace $\psi_j^{min,A^i,k}$ with $\psi_{j_{lightest}}^{min,A^i,k}$. This transformation does not destroy compatibility (since $\psi_{j_{lightest}}^{min,A^i,k}$ and $\psi_j^{min,A^i,k}$ have the same interval), nor does it change the weight of the set (since $\psi_{j_{lightest}}^{min,A^i,k}$ and $\psi_j^{min,A^i,k}$ have the same weight $w(i,k)$). Note that $\psi_{j_{lightest}}^{min,A^i,k} \in \overline{\Psi}$. Therefore, through a finite number of such transformations, we can transform the optimal solution $\Psi_{w,opt}$ to another optimal solution $\Psi_{w,opt'} \subseteq \overline{\Psi}$. □

Proof of Theorem 3: Algorithm 2 is the dynamic programming algorithm for weighted interval scheduling [8]. □

Proof of Theorem 4: Algorithm 3 starts with setting $w(\underline{i},\underline{k})$ to 0, the minimum possible value for $w(\underline{i},\underline{k})$. It then scans left-to-right to evaluate function values at chosen points.

Suppose that we are at a current value $x \geq 0$ of $w(\underline{i},\underline{k})$. We perform the dynamic programming algorithm for weighted interval scheduling to compute a set of matches for the interval scheduling problem of $\mathbb{M}(1:j)$ for $j = 1, 2, \ldots, M$.

In addition to computing the current set of intervals, we also check the trend when $w(\underline{i},\underline{k})$ is increased by an infinitesimal amount. For this purpose, we use $a[j]$ to denote the number of selected intervals for $\mathbb{M}(1:j)$ whose weight is $w(\underline{i},\underline{k})$. If we find $\mathbb{M}[j].w + OPT[p[j]] > OPT[j-1]$ in Line 8, $\mathbb{M}[j]$ is selected in the optimal solution for $\mathbb{M}(1:j)$. However, if the condition in Line 11 is also true, then the condition in Line 8 will no longer be true if $w(\underline{i},\underline{k})$ is increased from its current value of x to $x + \frac{\mathbb{M}[j].w + OPT[p[j]] - OPT[j-1]}{a[j-1] - a[p[j]] - \delta[j]}$ plus an infinitesimal amount. Therefore we add a corresponding upper-bound on σ in Line 12.

If we find $\mathbb{M}[j].w + OPT[p[j]] < OPT[j-1]$ in Line 8, $\mathbb{M}[j]$ is not selected in the optimal solution for $\mathbb{M}(1:j)$, and control goes to Line 14. However, if the condition in Line 16 is also true, then the condition in Line 8 will become true if $w(\underline{i},\underline{k})$ is increased from its current value of x to $x + \frac{\mathbb{M}[j].w + OPT[p[j]] - OPT[j-1]}{a[j-1] - a[p[j]] - \delta[j]}$ plus an infinitesimal amount. Therefore we add a corresponding upper-bound on σ in Line 17.

In Lines 19-24, we set the correct value of $a[j]$ to reflect the trend of change of $\mathbb{M}^w$ when $w(\underline{i},\underline{k})$ is increased from its current value of x by an infinitesimal amount.

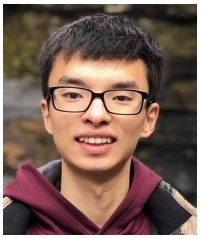
Algorithm 3 considers all possible cases of the upper bound. If the condition in Line 25 is true, we have reached the right end. Otherwise, we need to continue the left-to-right scan.

Given the discrete feature of the dynamic programming algorithm, we do not know whether the function is left continuous or right continuous at the boundary points. Therefore we also evaluate the function value at the mid-point of x and μ, which is ν. Since the function only takes $O(M)$ different values, the **goto** statement in Line 36 is executed $O(M)$ times. Since the execution of Line 2 to Line 35 uses $O(M)$ time, the algorithm has a worst-case time complexity $O(M^2)$. □

Proof of Theorem 5: Let $w^{[0]}$ be the initial weight vector. $f(w^{[0]})$ is a non-negative integer. Whenever the algorithm finds an improved solution, the objective function value is reduced by at least 1. Therefore the algorithm is guaranteed to terminate. When the algorithm terminates, it must be at a weight vector from which the objective function value cannot be improved by minimization over any of the N variables. This proves the theorem. □

Guoliang Xue (Member 1996, Senior Member 1999, Fellow, 2011) is a Professor of Computer Science at Arizona State University. His research interests span the areas of wireless networking, security and privacy, and optimization. He received the IEEE Communications Society William R. Bennett Prize in 2019. He has served as VP-Conferences of the IEEE Communications Society, and an editor for IEEE Transactions on Mobile Computing and IEEE/ACM Transactions on Networking. He is the Steering Committee Chair of IEEE INFOCOM.

Yinxin Wan (Student Member 2020) received his B.E degree in Information Security from University of Science and Technology of China in 2018. He is currently a Ph.D. student of Computer Science at Arizona State University. His research interests include cyber security, the Internet of Things, and data-driven networked systems.



Xuanli Lin (Student Member 2022) received his B.S. and M.S. degrees in Computer Science from Arizona State University in 2018 and 2020, respectively. He is currently a Ph.D. student of Computer Science at Arizona State University. His research interests include network optimization, machine learning, and the Internet of Things.

Kuai Xu (Member 2008, Senior Member 2015) is a Professor at Arizona State University. He received B.S. and M.S. degrees in Computer Science from Peking University, China in 1998 and 2001, and received Ph.D. degree in Computer Science from the University of Minnesota in 2006. His research interests include network security, Internet measurement, big data, data mining, and machine learning. He is a member of ACM and a senior member of IEEE.

Feng Wang (Member 2007) received the B.S. degree from Wuhan University in 1996, the M.S. degree from Peking University in 1999, and the Ph.D. degree from University of Minnesota, Twin Cities in 2005, all in Computer Science. She is currently a Professor with School of Mathematical and Natural Sciences, Arizona State University. Her research focuses on network science, social media analysis, network optimization, network security, and wireless sensor networks.