IEICE TRANS. INF. & SYST., VOL.E94–D, NO.3 MARCH 2011

LETTER

# Efficient Discovery of Highly Interrelated Users in One-Way Communications

**Jihwan SONG**[†a)], ***Student Member***, **Deokmin HAAM**[†], **Yoon-Joon LEE**[†], ***and*** **Myoung-Ho KIM**[†], ***Nonmembers***

**SUMMARY** In this paper, we introduce a new sequential pattern, the *Interactive User Sequence Pattern* (IUSP). This pattern is useful for grouping highly interrelated users in one-way communications such as e-mail, SMS, etc., especially when the communications include many spam users. Also, we propose an efficient algorithm for discovering IUSPs from massive one-way communication logs containing only the following information: senders, receivers, and dates and times. Even though there is a difficulty in that our new sequential pattern violates the *Apriori* property, the proposed algorithm shows excellent processing performance and low storage cost in experiments on a real dataset.

***key words:*** *data mining, sequential pattern, Apriori property violation*

## 1. Introduction

Recently, various one-way communication services (e.g., e-mail, SMS) have been used in connection with crime. To prevent, investigate, detect, or prosecute serious crimes that use such services, law enforcement agencies (e.g., police, FBI) are legally permitted to access electronic records of the communications, often called *logs*. The logs usually include senders, receivers, dates and times, and locations; however, communication content is excluded owing to privacy or technical issues[*]. Unfortunately, the process of manual log analysis is both time-consuming and labor-intensive because logs are usually massive and, moreover, include a lot of meaningless information generated by spam users.

To effectively find highly interrelated users from massive one-way communication logs, we introduce a new sequential pattern, the *Interactive User Sequence Pattern* (IUSP). First, an *Interactive User Sequence* (IUS) is a sequence of users that communicate interactively; for example, supposing a user $A$ sent a message to a user $B$ and then $B$ back to $A$, a user sequence $\langle AB \rangle$ can be formed from the interactive communication between $A$ and $B$. (The formal definitions with the time constraints and more details are given in Sect. 2.) The sequence $\langle AB \rangle$ is called an IUS. As shown Fig. 1 (a), if an IUS is derived from frequent interactive communications, it is called an IUSP. Here, frequent interactive communications mean that the frequency of communication is greater than or equal to a user-specified value. Generally, users included in an IUSP can be considered to be highly interrelated because frequent interactive communications between users indicates the close relationship of those users. In contrast, if a user unilaterally sends messages to others, as shown Fig. 1 (b), the recipients can hardly be said to be highly interrelated to the sender. Typical examples of this scenario are email or SMS spams; i.e., most users getting spam messages ignore them and might not reply to the spam senders.

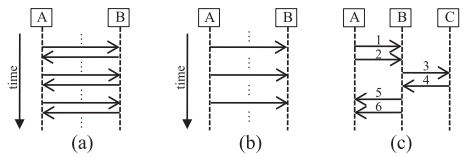


**Fig. 1** Examples of (a) an IUSP $\langle AB \rangle$ derived from frequent interactive communications, (b) non-interactive (unilateral) communication, and (c) the *Apriori* property violation for IUSs.

However, most existing sequential pattern mining algorithms [1]–[5] may not be able efficiently to discover IUSPs from the massive logs because the downward-closure property of sequential patterns, a.k.a. the *Apriori* property[**], does not hold for IUSs. In general, the *Apriori* property violation causes performance degradation because the existing algorithms directly or indirectly use the property to increase their performance. Figure 1 (c) shows an example of the *Apriori* property violation for IUSs. Depending on the counting scheme of applications, the IUS $\langle ABC \rangle$ has from at least one to at most four interactive communications; i.e., at most, it can be derived from the following four interactive communications: $A \rightleftarrows^{1}_{5} B \rightleftarrows^{3}_{4} C$, $A \rightleftarrows^{1}_{6} B \rightleftarrows^{3}_{4} C$, $A \rightleftarrows^{2}_{5} B \rightleftarrows^{3}_{4} C$, and $A \rightleftarrows^{2}_{6} B \rightleftarrows^{3}_{4} C$. (Here, $u_1 \rightleftarrows^{t_1}_{t_2} u_2$ means $u_1$ sent a message to $u_2$ at time $t_1$ and then $u_2$ back to $u_1$ at time $t_2$.) However, the IUS $\langle BC \rangle$, the sub-sequence of $\langle ABC \rangle$, has only one interactive communication $B \rightleftarrows^{3}_{4} C$. Hence, the *Apriori* property does not always hold for IUSs.

In this paper, we propose an efficient algorithm, the *IUSPMiner*, for discovering IUSPs from a massive one-way communication log. To improve performance, our algorithm reduces search space by pruning infrequent interactive communications of IUSs that do not violate the *Apriori* property in the middle of the process. To do this, IUSPMiner, first, finds some part that may cause a violation of the *Apri-*

Manuscript received May 25, 2010.

[†]The authors are with Division of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea.

a) E-mail: jhsong@dbserver.kaist.ac.kr

DOI: 10.1587/transinf.E94.D.714

[*]EU: Data Retention Directive (Directive 2006/24/EC); UK: Anti-Terrorism, Crime and Security Act 2001.

[**]If a sequence is frequent, then all of its sub-sequences must also be frequent.

*ori* property; then, the interactive communications related to the violation part are protected from being pruned. This is because IUSs derived from protected interactive communications may violate the *Apriori* property. Our performance study over a real dataset shows that IUSPMiner outperforms the Depth-First-Search (DFS)-based IUSPMiner, which is a variation of a DFS-based sequential pattern mining algorithm to discover IUSPs. Note that DFS-based sequential pattern mining algorithms such as *PrefixSpan* [4] show the best performance in many general cases.

## 2. Interactive User Sequence Patterns

This section formally defines an IUSP and its related terminology. First, a *communication event*, denoted by $(u_i, u_j, t_n)$, consists of two distinct users $u_i$ and $u_j$ and a date and time $t_n$, which means $u_i$ sent a message to $u_j$ at $t_n$. A *communication log* is a set of communication events and can be represented in the text-based (left) or graphic-based (right) form, as shown in Fig. 2 (a).

An *Interactive Communication Pair* (ICP) $p$, denoted by $(u_i, u_j, t_m, t_n)$, means a pair of communication events $(u_i, u_j, t_m)$ and $(u_j, u_i, t_n)$ such that $0 < t_n - t_m \leq W_{max}$. Here, $u_i$ denoted by $SRC(p)$, $u_j$ by $DST(p)$, $t_m$ by $ST(p)$, and $t_n$ by $ET(p)$ are called the *source*, *destination*, *starting time*, and *ending time* of $p$, respectively. Also, $t_n - t_m$ denoted by $RT(p)$ and $[t_m, t_n]$ by $TI(p)$ mean the *response time* and *time interval* of $p$, respectively. Especially, $W_{max}$ is a user-specified *maximum response deadline*; i.e., we assume that the two communication events $(A, B, t_1)$ and $(B, A, t_2)$ are totally unrelated to each other if $|t_2 - t_1| > W_{max}$. For $W_{max} = 20$, Fig. 2 (b) shows all ICPs in the communication log of (a).

An *Interactive Communication Sequence* (ICS), denoted by $\langle p_0\ p_1\ \ldots p_{n-1}\rangle$, is a sequence of $n$ ICPs $p_0, p_1, \ldots, p_{n-1}$ such that (1) $DST(p_i) = SRC(p_{i+1})$, (2) $SRC(p_i) \notin \bigcup_{k=i+1}^{n-1}\{SRC(p_k)\} \cup \{DST(p_{n-1})\}$, (3) $TI(p_i)$ completely includes $TI(p_{i+1})$, and (4) $RT(p_i) \leq UpperRespDeadLine(p_{i+1}) + \alpha$, where $RT(p_{n-1}) \leq \alpha$ and $UpperRespDeadLine(p) = \alpha \times \lceil \frac{RT(p)}{\alpha} \rceil$. Here, $\alpha$ is a user-specified *two-party response deadline*; i.e., we assume $p_i$ and $p_{i+1}$ are related to each other if $RT(p_i) \leq UpperRespDeadLine(p_{i+1}) + \alpha$, where $RT(p_{n-1}) \leq \alpha$. When $\alpha = 5$, all ICSs from the ICPs of (b) are shown in Fig. 2 (c).

Lastly, an *Interactive User Sequence* (IUS), denoted by $\langle u_0\ u_1\ \ldots u_{n-1}\rangle$, is a sequence of $n$ users $u_0, u_1, \ldots, u_{n-1}$ such that at least one ICS $\langle p_0\ p_1\ \ldots p_{n-2}\rangle$ exists, where $SRC(p_0) = u_0$, $SRC(p_1) = u_1, \ldots, SRC(p_{n-2}) = u_{n-2}$, and $DST(p_{n-2}) = u_{n-1}$. Since an IUS can be derived from one or more ICSs, the frequency of an IUS is defined as the number of its ICSs; i.e., the *frequency* of an IUS $u$, denoted by $Freq(u)$, is $k$ if $u$ is derived from $k$ ICSs. An *Interactive User Sequence Pattern* (IUSP) is an IUS $u$ such that $Freq(u) \geq minsup$. Here, *minsup* is a user-specified *minimum support*. Figure 2 (d) shows all IUSs in the ICSs of (c) with their frequencies. (The number after the ":" sign indicates the frequency of corresponding IUSs.) If $minsup = 3$, IUSs $\langle C\ D\rangle$, $\langle B\ C\ D\rangle$, and $\langle A\ B\ C\ D\rangle$ become IUSPs, since only these IUSs have three or more ICSs.

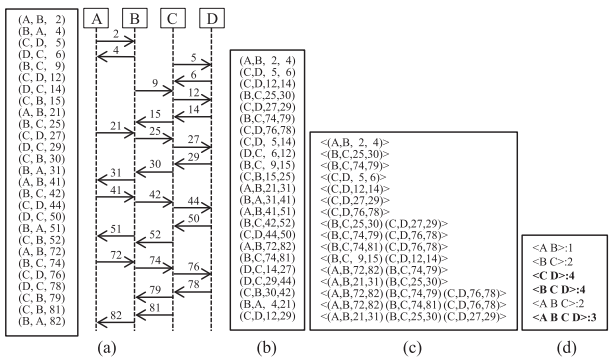


**Fig. 2** (a) a communication log: text-based (left) and graphic-based (right) representation, (b) ICPs, (c) ICSs, and (d) IUSs and their corresponding supports.

## 3. IUSP Mining

We propose an algorithm, the *IUSPMiner*, to efficiently discover IUSPs from the massive communication log. IUSPMiner consists of three phases: (1) *Overlap Detection*, (2) *ICSFrag Protection*, and (3) *IUSP Generation*.

### 3.1 Overlap Detection

The first phase is to detect *overlaps* between ICPs, which are likely to cause violations of the *Apriori* property. First, the algorithm finds all the ICPs from the given communication log and then divides them into groups called *ICPGroups* according to their response times. Given $W_{max}$ and $\alpha$, $n$ ICPGroups, $g_0, g_1, \ldots, g_{n-1}$, are created such that $g_i$ has ICPs whose response times are between interval $(i\times\alpha, (i+1)\times\alpha]$, where $n = \lceil \frac{W_{max}}{\alpha} \rceil$. Here, $i$ is called a *group number* of $g_i$, denoted by $GN(p)$, where $p \in g_i$. After the ICP discovery, the algorithm finds overlaps between ICPs. As shown in Fig. 3, two ICPs $p$ and $p'$ *overlap* one another if the following conditions hold: (1) $TI(p)$ overlaps with $TI(p')$, (2) $SRC(p) = SRC(p')$ and $DST(p) = DST(p')$, (3) $|GN(p) - GN(p')| \leq 1$, and (4) $MAX(GN(p), GN(p')) > 0$. For each of the overlaps between ICPs $p$ and $p'$, the algorithm forwards a tuple $(o, M, d)$ called *ICP Overlap Information* (IOInfo) to the next phase, where $o$ is an *overlapped time interval* (OTI) $[Max(ST(p), ST(p')), Min(ET(p), ET(p'))]$,

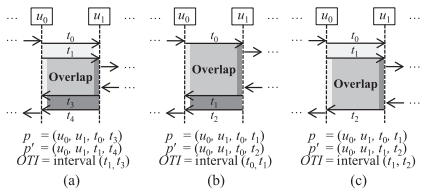


**Fig. 3** Overlaps between ICPs, where $|GN(p) - GN(p')| \leq 1$ and $MAX(GN(p), GN(p')) > 0$.

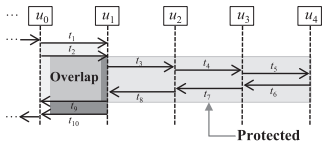


**Fig. 4** A protected ICSFrag $\langle u_1\ u_2\ u_3\ u_4 \rangle$ when ICPs $(u_0, u_1, t_1, t_9)$ and $(u_0, u_1, t_2, t_{10})$ overlap one another.

$M$ is $MAX(GN(p), GN(p'))$, and $d$ is $DST(p)$ or $DST(p')$.

### 3.2 ICSFrag Protection

In the second phase, the algorithm *protects* some *ICS fragments* (ICSFrags) using the *IOInfo* forwarded from the first phase. Here, ICSFrags can be part of ICSs if several conditions hold, and, particularly, ICSs including *protected* ICSFrags are likely to cause *Apriori* property violation. Suppose that $g_i$ is a non-empty ICPGroup, an ICSFrag $f$ is an ordered list of ICPs, $\langle p_0\ p_1\ \ldots p_{n-1} \rangle$, where $p_j \in g_i$, if the following conditions hold: (1) $n = 1$ or (2) $n > 1$, $DST(p_j) = SRC(p_{j+1})$, $TI(p_j)$ completely includes $TI(p_{j+1})$, and $SRC(p_j) \notin \bigcup_{k=j+1}^{n-1}\{SRC(p_k)\} \cup \{DST(p_{n-1})\}$. Here, $p_0$ is denoted by $FirstICP(f)$ and $i$ is called a group number of $G_i$, denoted by $GN(f)$, where $f \in G_i$. First, the algorithm finds all ICSFrags from each of the non-empty ICPGroups and then divides them into groups called *ICSFragGroups* according to their origins; i.e., ICSFragGroup $G_i$ has only the ICSFrags discovered from ICPGroup $g_i$. After the discovery of ICSFrags, as shown Fig. 4, for each tuple (o, M, d) of all $IOInfo$, the algorithm protects an ICSFrag $f$ such that $TI(FirstICP(f))$ is completely included in $o$, $|GN(f) - M| \leq 1$, and $d = SRC(FirstICP(f))$.

### 3.3 IUSP Generation

Lastly, the algorithm merges ICSs and ICSFrags to make new ICSs and derives IUSPs from the newly created ICSs, as shown in Fig. 5. In the figure, $l$-ICSs and $l$-ICSFs mean ICSs and ICSFrags with length $l$; i.e., they all consist of $l$ ICPs. First, the algorithm makes 1-ICSs from 1-ICSFrags in ICSFragGroup $G_0$, since 1-ICSFrags themselves in $G_0$ are all 1-ICSs, that is, they satisfy all the conditions for ICSs. Then, the algorithm derives 1-IUSs from the 1-ICSs and picks out IUSPs. At this point, the algorithm prunes unnecessary ICSs that both are not deduced to IUSPs and do not include protected ICSFrags, which removal leads to performance improvement. To generate $l$-ICSs ($l > 1$), the algorithm merges $j$-ICSFrags in $G_i$ with $(l - j)$-ICSs, where $0 < j < l$ and $0 < i < MIN(l, m)$. After generating $l$-ICSs, the algorithm derives IUSPs and prunes unnecessary ICSs. This process is repeated until no more ICSs are newly created. Figure 6 describes the pseudo code of Algorithm *IUSP_Generation*. Note that procedure *DeriveAndPrune(temp, ResultSet, minsup)* in Step 21 performs deriving of IUSs and pruning of unnecessary ICSs as follows: (1) it derives IUSs from *temp* that stores ICSs, (2) it adds IUSPs to $ResultSet$, and (3) it prunes unnecessary ICSs.

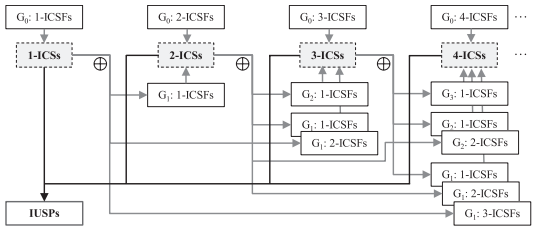


**Fig. 5** Merging ICSs and ICSFrags and deriving IUSPs.

**Algorithm** *IUSP_Generation*

**Input:**
*minsup*: a minimum support threshold
$G_0, G_1, \ldots, G_{m-1}$: ICSFragGroups

**Output:**
*ResultSet*: a set for IUSPs

**Description:**
01: *ResultSet* ← {}, *ICSSet* ← {}
02: $l \leftarrow 1$
03: **repeat**
04: *temp* ← {}
05: Generate *l-ICSs* from *l-ICSFrags* in $G_0$ and add the *l-ICSs* into *temp*
06: $i \leftarrow 1$
07: **while** $i <$ MIN($l$, $m$) **do**
08: **if** $i < l$ -1 **then**
09: $j \leftarrow 1$
10: **while** $j < l$ **do**
11: Generate *l-ICSs* by merging *j-ICSFrags* in $G_i$ with $(l-j)$-*ICSs* in *ICSSet*
12: Add the *l-ICSs* into *temp*
13: $j \leftarrow j + 1$
14: **end while**
15: **else**
16: Generate *l-ICSs* by merging 1-*ICSFrags* in $G_i$ with (*l*-1)-*ICSs* in *ICSSet*
17: Add the *l-ICSs* into *temp*
18: **end if**
19: $i \leftarrow i + 1$
20: **end while**
21: *DeriveAndPrune* (*temp*, *ResultSet*, *minsup*)
22: *ICSSet* ← *ICSSet* ∪ *temp*
23: $l \leftarrow l + 1$
24: **until** no more newly generated *ICSs*

**Fig. 6** IUSP_Generation algorithm.

## 4. Experiments

In the experiments, we compared IUSPMiner with a *Depth-First-Search (DFS)-based IUSPMiner* (DFS-IUSPMiner). Since no previous work for discovering IUSPs exists, we made the DFS-IUSPMiner find IUSPs by modifying an existing sequential pattern mining algorithm based on depth-first-search, generally following the pattern of the best-performing algorithm [4] in sequential pattern mining; it works like a DFS-based sequential pattern mining algorithm but does not prune any ICSs that are not deduced to IUSPs because of the *Apriori* property violation of IUSPs.

We created a real communication log generated from the Enron e-mail dataset [6], which is a rich source of information showcasing the internal working of an actual corporation over the period of 1998 to 2002. In the midst of Enron's legal troubles in 2002, the Federal Energy Regulatory Commission in the United States made the dataset available to the public. The communication log contain at most 650,000 communication events as generated from the information of senders, receivers, and dates and times of the Enron e-mail dataset between 1 Oct. 2000 and 30 Sep. 2001. To compare the performance of the two algorithms, we measured their running times and peak memory usages. All experiments were repeated ten times, and the average was taken.

First, we present the results of scale-up experiments;

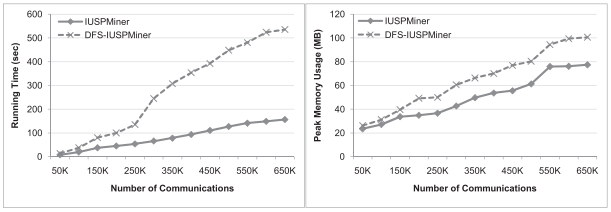


**Fig. 7** Running times (left) and peak memory usages (right) with variation of the size of communication log.

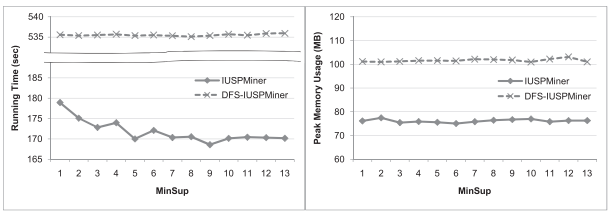


**Fig. 8** Running times (left) and peak memory usages (right) with varying of *minsup*.

i.e., the number of communication events was increased thirteen times from 50,000 to 650,000. Here, $\alpha$ was set to 14,400 s (4 hours), $W_{max}$ to 172,800 s (2 days), and *minsup* to 20. Figure 7 shows the running times (left) and peak memory usages (right) of the two algorithms while varying the size of the communication log. At 50,000 communication events, both algorithms showed similar performance. However, by increasing the number of communication events, the running time of DFS-IUSPMiner increased more rapidly than that of IUSPMiner; for 650,000 communication events, IUSPMiner was about 3.42 times faster than DFS-IUSPMiner. In the case of peak memory usages of the two algorithms, IUSPMiner consumed less memory than DFS-IUSPMiner. This is because IUSPMiner can prune unnecessary ICSs, whereas DFS-IUSPMiner cannot.

Also, we compared two algorithms with *minsup* varied from one to thirteen. Here, 650,000 communication events were used, and $\alpha$ and $W_{max}$ were set to 14,400 s and 172,800 s, respectively. Figure 8 shows the running times (left) and peak memory usages (right) of the two algorithms with varying of *minsup*. The running time of IUSPMiner was decreased as *minsup* increased while DFS-IUSPMiner kept similar running times with any *minsup*. This improvement is because of the protecting and pruning systems of IUSPMiner. However, the peak memory usages of two algorithms were nearly unchanged.

From the results so far, we can confirm that IUSPMiner needed less time and consumed less memory than DFS-IUSPMiner.

## 5. Conclusion

This paper introduces a new sequential pattern called an IUSP to group highly inter-related users in one-way communications. The distinct characteristic of the new pattern is to violate the *Apriori* property. This feature leads to lower performance for existing sequential pattern mining algorithms for discovering IUSPs, since unnecessary intermediate results cannot be pruned. Our proposed algorithm IUSPMiner overcomes this difficulty by protecting part of the *Apriori* property violation; i.e., the algorithm prunes unnecessary intermediate results that both are not deduced to frequent patterns and do not include protected parts. This new protecting and pruning strategy yields better performance and will help other algorithms mine some sequence patterns that disobey the *Apriori* property.

## Acknowledgments

This work was supported (i) by the 2007 ETRI contract research, titled “Analyses of User Communication Patterns in Wireless Internet”, and (ii) by Basic Science Research Program through the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2009-0089128).

### References

[1] R. Agrawal and R. Srikant, “Mining sequential patterns,” Proc. 11th ICDE, Taipei, Taiwan, 1995.
[2] H. Mannila, H. Toivonen, and A.I. Verkamo, “Discovery of frequent episodes in event sequences,” Data Mining and Knowledge Discovery, vol.1, no.3, pp.259–289, 1997.
[3] M.J. Zaki, “SPADE: An efficient algorithm for mining frequent sequences,” Mach. Learn., vol.42, no.1/2, pp.31–60, 2001.
[4] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, “Mining sequential patterns by pattern-growth: The PrefixSpan approach,” IEEE Trans. Knowl. Data Eng., vol.16, no.11, pp.1424–1440, 2004.
[5] S. Aseervatham, A. Osmani, and E. Viennet, “bitSPADE: A lattice-based sequential pattern mining algorithm using bitmap representation,” Proc. 6th ICDM, Hong Kong, China, 2006.
[6] J. Shetty and J. Adibi, “The Enron e-mail dataset database schema and brief statistical report,” Technical Report, Information Sciences Institute, 2004.