대화형 통신 순서열 패턴의 마이닝을 위한 효율적인 알고리즘

(An Efficient Algorithm for Mining Interactive Communication Sequence Patterns)

함 덕 민 † 송 지 환 † 김 명 호 ††
(Deokmin Haam) (Jihwan Song) (Myoung Ho Kim)

요 약 통신 기록 데이타는 이메일이나 인스턴스 메시지를 주고 받거나, 웹사이트에 접속하는 것과 같은 통신 이벤트들로 구성된다. 미국과 유럽연합을 포함한 여러 나라에서는 인터넷을 사용한 범죄의 조사와 발견을 위해서 통신 서비스 제공자에게 이런 데이타를 보관하도록 규정하고 있다. 보관되는 통신 기록 데이타의 크기가 매우 크기 때문에 치안당국이 이 데이타를 사용하기 위해서는 필요한 정보만을 효과적으로 추출해내는 방법이 필요하다. 본 논문에서는 발신자, 수신자, 통신발생시각의 세 가지 정보만 포함하는 통신 이벤트가 주어질 때, 의미 있는 정보 중 하나인 대화형 통신 순서열 패턴과 이러한 패턴의 마이닝 문제를 정의하고 그것을 해결하기 위해 Fast Discovering Interactive Communication Sequence Patterns (FDICSP)라 불리는 알고리즘을 제안한다. FDICSP는 길이가 짧은 대화형 통신 순서열을 조합하여 길이가 긴 대화형 통신 순서열을 생성해나가는데, 대화형 통신 순서열의 특성에 초점을 맞춘 작업을 통해 효율적으로 대화형 통신 순서열 패턴을 찾는다.

키워드 : 데이타 마이닝, 순서열 패턴 마이닝, 대화형 통신 순서열, 통신 기록 보관

Abstract Communication log data consist of communication events such as sending and receiving e-mail or instance message and visiting web sites, etc. Many countries including USA and EU enforce the retention of these data on the communication service providers for the purpose of investigating or detecting criminals through the Internet. Because size of the retained data is very large, the efficient method for extracting valuable information from the data is needed for Law Enforcement Authorities to use the retained data. This paper defines the Interactive Communication Sequence Patterns(ICSPs) that is the important information when each communication event in communication log data consists of sender, receiver, and timestamp of this event. We also define a Mining(ICSPM) problem to discover such patterns and propose a method called Fast Discovering Interactive Communication Sequence Pattern(FDICSP) to solve this problem. FDICSP focuses on the characteristics of ICS to reduce the search space when it finds longer sequences by using shorter sequences. Thus, FDICSP can find Interactive Communication Sequence Patterns efficiently.

Key words : Data mining, Sequential pattern mining, Interactive communication Sequence, telecommunications data retention

· 이 논문은 2007년도 정부(교육과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(No. R0A-2007-000-10046-0)

† 학생회원 : KAIST 전산학과
dmhaam@gmail.com
jhsong@dbserver.kaist.ac.kr
†† 종신회원 : KAIST 전산학과 교수
mhkim@dbserver.kaist.ac.kr

논문접수 : 2008년 6월 23일
심사완료 : 2009년 2월 12일

정보과학회논문지: 데이타베이스 제36권 제3호(2009.6)

1. 서 론

순서열 패턴 마이닝(Sequential pattern mining)은 순서열 데이타베이스에서 빈번하게 존재하는 부분 순서열(sub-sequence)들을 찾는 데이타 마이닝 문제이다. 이러한 순서열 패턴 마이닝은 소비자의 구매 패턴 분석이나 DNA 순서열 분석 등과 같은 다양한 분야에서 사용되고 있다.

순서열 패턴 마이닝 방법 중 초기에 연구된 방법은 *Apriori* 특성을 활용한 방법들이다. *Apriori* 특성 이란 길이가 긴 순서열은 그 순서열을 생성하는데 사용된 짧

은 길이의 순서열 보다 그 빈도수가 적게 되는 특성이다. 이 방법들은 기본적으로 길이가 k인 순서열들을 찾은 후, 그들을 조합하여 길이가 k+1인 순서열을 생성하는 방식으로 진행된다. 길이가 1인 순서열을 찾는 것에서 시작하여, 더 이상 새로운 순서열이 생성되지 않을 때까지 같은 작업을 반복하는데, *Apriori* 특성을 활용하여 각 단계에서 빈번하지 못한 순서열을 제거함으로써 다음 단계에서 고려해야 하는 순서열의 수를 줄여서 탐색시간을 줄이는 것이 이 방법들의 특징이다.

다른 순서열 패턴 마이닝 방법으로 Pattern Growth에 기반한 방법들이 제안되었다. 이 방법들은 순서열 데이타베이스를 분할하고 각 부분에서 빈번한 패턴을 찾는 작업을 재귀적으로 수행해 나가면서, 최종적으로 모든 빈번한 순서열을 찾는다. 이 방법 역시 도중에 생성되는 빈번하지 않는 순서열을 제거해서 탐색시간을 줄이는 방식을 사용한다.

기존의 순서열 패턴과는 다른 새로운 종류의 순서열 패턴의 등장에 따라 새로운 순서열 패턴 마이닝 방법이 필요하게 되었다. 새로운 종류의 순서열 패턴의 한 예로 통신 기록 보관(Telecommunications Data Retention)을 들 수 있다. 통신 기록 보관이란 국가의 통신 서비스들의 합법적인 감청을 쉽게 하기 위해 일정 기간 동안 전화의 통화 기록(CDR: Call Detail Records)이나 인터넷 기반의 서비스 사용 기록(IPDR: Internet Protocol detail records)을 저장하는 것을 말한다. 즉, 전화를 누가 누구에게 언제 걸고 받았으며, 이메일을 누가 누구에게 언제 보냈으며, 웹 페이지에 누가 언제 접근하였는지에 대한 기록들을 일정 기간 저장하는 것이다. 일반적으로, 통신 내용은 통신 기록 보관에서 제외된다. 2006년 3월 유럽연합은 모든 회원국들은 *Directive 2006/24/EC*에 명시되어 있는 필요한 데이타들을 6개월에서 2년 동안 저장하는 것을 공식적으로 의무화하였다[1]. 비슷하게, 여러 다른 나라들 역시 통신 기록 보관을 법제화하였다. 이러한 저장된 통신 기록은 사법 집행 기관이 범죄 수사에 이용할 수 있다[2-4].

본 논문에서는 저장된 통신 기록에서 추출해 낼 수 있는 유용한 정보 중 하나인 대화형 통신 순서열(ICS: Interactive Communication Sequence)을 효율적으로 찾는 방법을 제안한다. 그리고 ICS 중에서 빈번하게 발생하는 것은 대화형 통신 순서열 패턴(ICSP: Interactive Communication Sequence Pattern)이라고 정의하는데, 이것은 통신 사용자간에 발생한 의미 있는 통신의 흐름을 의미한다. ICSP를 찾게 되면, 서로 밀접하게 연락을 주고받는 사용자의 그룹을 찾아내거나, 그 그룹 내에서 통신이 전달되는 흐름을 파악할 수 있게 된다. 예를 들어, 특정 국가에서 테러에 대처하기 위해 통신 기록을 수집하여 그것에서 연관된 사용자 그룹을 파악해 낸다면, 소수의 테러리스트에 대한 정보와 ICSP를 통해 얻어낸 사용자 그룹 정보를 통해 테러와 연관된 많은 수의 테러리스트를 추정해 낼 수 있다. 또한, 그들 사이의 통신의 흐름을 파악함으로써 그 조직 내부의 의사결정 단계 및 특성을 추측해내면, 현재 진행되는 테러 음모가 어떤 시점(준비단계, 실행에 착수한 상태)에 도달해 있는 지 등을 파악하여 그에 대한 대책을 세우는 데 활용할 수 있을 것이다.

1.1 관련 연구

Apriori 특성을 사용한 순서열 패턴 마이닝 중 하나인 [5]에서는 이벤트 간의 부분 순서 집합을 에피소드(episode)로 정의하고, 순서열에 존재하는 에피소드 중에서 빈번하게 발생한 에피소드를 찾는다. 일단 입력 받은 순서열에서 정해진 타임윈도우 범위 안에 존재하는 모든 에피소드를 찾는다. 그 후 타임윈도우를 순서열을 따라 이동시키면서 에피소드를 찾는 작업을 반복한다.

[6]에서는 사용자가 정한 *support* 하한값보다 많이 나타나는 모든 순서열을 찾는다. 길이가 k-1인 모든 순서열을 먼저 찾아내고, 그 중에서 두 개를 조합하여 길이가 k인 순서열을 만들어 내는 방식으로 모든 길이의 순서열을 찾아낸다. 이 때, 길이가 k-1인 순서열 중에서 빈번하지 않은 순서열이 조합되어 생성되는 길이 k 인 순서열도 빈번하지 않다는 *Apriori* 특성[7]을 이용한다. 길이 k-1인 순서열 중에서 빈번하지 않는 것은 미리 제거한 후, 길이 k인 순서열을 조합함으로써 불필요한 작업을 줄인다.

[8]은 [6]의 방법처럼 *Apriori* 특성을 사용하여 점차 길이가 긴 순서열을 찾아나간다. 하지만 순서열 데이타베이스를 수직형 데이타 포멧(vertical data format) 형태로 표현하고, 조인을 통해 순서열을 조합하는 방법을 사용하여 순서열 데이타베이스에 대한 탐색 수를 줄인다. [9]는 [8]의 방법에 비트맵(bitmap)을 사용하여 성능이 향상된 방법을 제안했다.

Aprioi 특성을 사용하여 순서열 패턴을 찾아내는 위의 네 방법[5,6,8,9]과 달리 [10]과 [11]은 *Pattern Growth* 방법을 사용한 *PrefixSpan* 알고리즘이다. 이 알고리즘은 순서열 데이타베이스를 탐색하여 *support* 하한값보다 빈번하게 나타나는 길이 1인 모든 순서열을 찾아낸다. 그 후, 순서열 데이타베이스를 분류하는데 그 기준은 각 순서열이 앞에서 찾아낸 길이 1인 순서열 중 어떤 것으로 시작하는지가 된다. 예를 들면, 길이 1인 빈번한 순서열이 <a>, <c>일 때, 순서열 데이타베이스의 <a, (abc)>는 <a>로 프로젝트된 데이타베이스(<a>-projected DB)로 분류되고, <(cd),c>는 <c>로 프로젝트된 데이타베이스로 분류된다. 분류된 각 데이타

베이스에 대해 위 과정을 재귀적으로 수행하는 것으로 빈번한 순서열을 찾는다. *Pattern Growth* 방법은 *Apriori* 특성을 사용한 방법보다 적은 수의 탐색을 하는 장점이 있지만, 많은 수의 프로젝트된 데이타베이스(projected DB)를 생성하기 때문에 메모리 소비가 크다는 단점이 있다.

다차원 순서열 패턴을 마이닝하는 연구도 진행되었는데, [12]는 d차원의 순서열 데이타에서 *Apriori* 방법과 *Pattern Growth* 방법을 사용하여 빈번한 순서열을 찾는 방법을 제안했고, [13]은 데이타 스트림 환경에서 다차원 순서열 패턴을 찾는 방법을 제안한다. 그리고 [14]에서는 각 차원마다 구성요소 사이에 계층이 존재하는 환경에서의 다차원 순서열 패턴 마이닝 방법을 제안했다.

ICS의 특성을 고려하면, 기존의 순서열 패턴 마이닝 알고리즘으로 ICS를 찾는 것은 비효율적이다. 첫 번째 특성은 ICS의 순서열 패턴이 A→B→C→B→A나 A→C→A 처럼 대칭형이라는 것이다. A→B→C→B나 B→C→D→A→B 처럼 비 대칭형 순서열은 ICS가 되지 않는다. 기본적으로 순서열 패턴 마이닝 알고리즘에서는 찾는 순서열의 대칭성 여부는 고려하지 않고 모든 순서열을 찾아야 하기 때문에, 중간 과정에서 ICS와 무관한 비 대칭형 순서열 검색도 수행된다. 그러므로 대칭형 순서열에 초점을 맞춰서 검색하는 경우보다 비효율적이다.

두 번째 특성은 ICS는 타임윈도우가 일정하지 않다는 것이다. ICS는 순서열 패턴이 길어지면, 그 길이에 따라서 타임윈도우가 확장되지만, 순서열 패턴 마이닝 알고리즘에서는 하나의 고정된 타임윈도우를 사용한다. 다양한 크기의 타임윈도우를 가진 ICS를 모두 찾기 위해서는 ICS의 타임윈도우 중 가장 크기가 큰 것을 알고리즘의 타임윈도우로 정해야 한다. 순서열 패턴 마이닝 알고리즘은 타임윈도우가 커질수록 찾는 순서열의 수가 급격히 늘어나기 때문에, 수많은 불필요한 탐색이 발생한다.

세 번째 특성은 ICS 패턴은 일반적인 순서열 패턴과 달리 길이가 k인 순서열이 그것을 구성하는 길이가 k-1인 순서열보다 더 많이 나타날 수 있다는 것이다. 그렇기 때문에 기존의 순서열 패턴 마이닝 알고리즘들이 사용하는 *support* 하한값을 사용한 제거(pruning)를 적용할 경우 찾을 수 없는 ICS 패턴이 존재하기 때문에 기존 방법들은 *support* 하한값을 1로 고정해야 한다. 이 경우에 앞서 언급한 방법들은 심한 성능저하를 나타낸다.

본 논문은 통신 기록에서 추출해 낼 수 있는 유용한 정보 중 하나인 대화형 통신 순서열 패턴을 정의하고, 이러한 패턴의 마이닝 문제를 소개한다. 그리고, 순서열 패턴 마이닝 알고리즘에 적합하지 않는 ICS 검색을 효율적으로 수행할 수 있도록 ICS의 특성을 고려하여 불필요한 탐색을 최소화 하는 방법을 제안한다. 또한 큰 데이타 셋에서도 제안하는 방법이 큰 성능저하가 없음을 보이고, 대화형 통신 순서열을 찾는 직관적인 방법 및 기존방법의 개념을 적용하여 만든 *Exhaustive(Apriori-like)*, *Exhaustive(PatternGrowth-like)*과 비교하여 제안하는 방법의 효율성을 검증한다.

1.2 논문의 구성

본 논문의 구성은 다음과 같다. 2장에서는 대화형 통신 순서열 패턴 마이닝 문제를 정의하고, 3장에서 대화형 통신 순서열 패턴 마이닝의 효율적인 방법을 제안한다. 4장에서는 제안된 알고리즘의 성능을 실험을 통해 보여주고, 그 의미를 분석한다. 5장에서 본 연구의 결과를 정리하고 요약한다.

2. 대화형 통신 순서열(Interactive Communication Sequence)

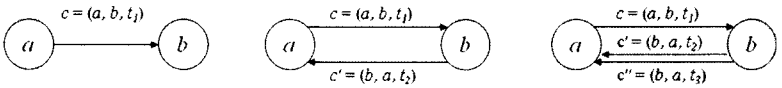
이 장에서는 대화형 통신 순서열(ICS: Interactive Communication Sequence)과 대화형 통신 순서열 패턴(ICSP: Interactive Communication Sequence Pattern)의 마이닝 문제를 정의한다.

정의 1. 통신(*communication*)은 (a, b, t)로 나타낸다. 이 때, a, b는 사용자 집합에 속하는 임의의 사용자이고, t는 통신 발생 시각이다.

통신 $c = (a, b, t)$는 사용자 a가 시각 t에 사용자 b에게 메시지를 보낸 통신 이벤트를 의미한다. 이 때, 통신의 발신자 $sender(c)$는 a, 통신의 수신자 $receiver(c)$는 b, 발생시각 $time(c)$는 t를 의미한다. 그림 1(a)는 사용자 a가 시각 t_i에 사용자 b에게 메시지를 보내는 통신을 나타낸다.

두 통신 c와 c'이 $sender(c) = receiver(c')$, $receiver(c) = sender(c')$이고, $| time(c') - time(c) | \leq W_{max}$일 때 c와 c'은 서로 역통신(*inverse*)라고 한다. 이 때, 최대 타임윈도우(*maximum time window*) W_{max}는 $| time(c') - time(c) | > W_{max}$ 일 경우 c와 c'의 내용은 서로 무관하다고 단정하는 기준값이다. $time(c')$이 $time(c)$보다 크고 서로 역통신 관계인 c와 c'의 순서쌍 (c, c')을 인버스 페어(Inverse Pair)라고 하고, $InvP(c, c')$로 표기한다. 그림 1(b)는 서로 역통신 관계인 두 통신 c와 c'이 구성하는 인버스 페어 $InvP(c, c')$를 나타낸다. 하나의 통신에 대해 여러 개의 역통신이 존재할 수 있는데, 그림 1(c)는 통신 c에 대해 두 개의 역통신 c'과 c''이 존재하는 경우를 나타낸다. 통신 c는 각각의 역통신과 인버스 페어 $InvP(c, c')$과 $InvP(c, c'')$를 구성하게 된다.

통신들은 컴퓨터 저장장치에 통신기록(*communication log*)의 형태로 저장된다. 통신기록 내의 통신들은 발생시각을 기준으로 오름차순으로 정렬되어서 보관된다.



(a) A communication (b) Inverse pair (c) multiple inverses

$(t_1 < t_2 < t_3, t_2 - t_1 < W_{max}$ and $t_3 - t_1 < W_{max})$

그림 1 통신과 역통신

정의 2. W_{max}가 주어지고 $W < W_{max}$인 타임윈도우 W가 주어졌을 때, 통신 c와 그 통신의 역통신 c'이 $0 < time(c') - time(c) \leq W$를 만족하면, 순서쌍 (c, c')은 대화형 통신(IC: Interactive Communication)이 된다. IC인 순서쌍 (c, c')은 IC(a, b)로 표기하는데, 두 파라메터(parameter) a와 b는 각각 *sender*(c)와 *receiver*(c)를 의미한다.

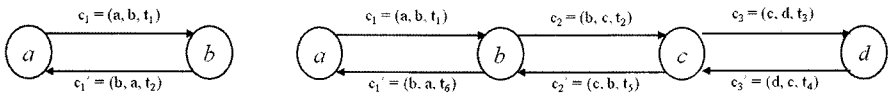
정의 2는 한 통신 c가 발생한 후, 그 통신의 역통신 c'이 주어진 타임윈도우 이내에 발생한 경우 인버스 페어 InvP(c, c')이 대화형 통신이 되는 것을 의미한다. 대화형 통신 IC(a, b)는 사용자 a로부터 메시지를 받은 사용자 b가 사용자 a에게 메시지를 보내는데, 그 내용이 사용자 a로부터 받은 메시지에 대한 응답일 가능성이 높음을 의미한다. 그림 2(a)의 두 통신 c, c'의 발생시각 차이가 주어진 타임윈도우보다 작거나 같은 경우(즉, $t_2 - t_1 \leq W$), (c, c')은 IC(a, b)가 된다.

그림 2(b)의 6개의 통신에 대해 W_{max}와 $W < W_{max}$인 타임윈도우 W가 주어졌을 때, $t_1 < t_2 < t_3 < t_4 < t_5 < t_6$ 이고 $t_6 - t_1 \leq W$이면, 통신 순서열 $<c_1, c_2, c_3, c_3', c_2', c_1'>$은 대화형 통신 순서열(ICS: Interactive Communication Sequence)이 된다. 이 때, 대화형 통신 순서열은 ICS(a, d)로 표기하고, a는 *source*(ICS(a, d)), d는 *destination*(ICS(a, d))가 된다. ICS의 *source*는 통신 순서열의 첫 통신을 시작하는 사용자를 의미하고, *destination*은 통신 순서열 상에서 source로부터 가장 먼 위치의 사용자를 의미한다. 그리고, 그림 2(b)의 통신 순서열이 ICS(a, d)라는 것은, 그 통신 순서열이 사용자 a가 사용자 b에게 보낸 메시지와 관련 있는 메시지들이 사용자 c, 사용자 d로 전달되고, 다시 사용자 c, 사용자 b, 사용자 a로 응답되는 형태로 돌아오는 통신일 가능성이 높음을 의미한다.

ICS(a, d)의 조건에 의해 $t_2 < t_3 < t_4 < t_5$ 이고 $t_5 - t_2 \leq W$ 이므로 통신 순서열 $<c_2, c_3, c_3', c_2'>$도 대화형 통신 순서열 ICS(b, d)가 된다. 그러므로 그림 2(b)의 통신 순서열 $<c_1, c_2, c_3, c_3', c_2', c_1'>$은 통신 순서열 $<c_1$, ICS(b, d), $c_1'>$로 나타낼 수 있고, 이것은 ICS(a, d)를 의미한다. 즉, ICS는 다른 ICS를 포함할 수 있는데, 포함되는 ICS를 *sub-ICS*라고 한다. 그림 2(b)에서 ICS(b, d)는 ICS(a, d)의 *sub-ICS*이다. 그리고 ICS $<c_1, c_2, \cdots, c_n, c_n', \cdots, c_2', c_1'>$의 길이는 $n-1$로 정의한다. ICS의 길이는 그 ICS 내에서 최초로 응답 메시지가 전송되기 전까지 수행된 통신의 개수와 동일하다. 예를 들면, 그림 2(b)의 ICS(a, d)의 길이는 3이 된다. 그리고 길이가 1인 ICS는 서로 역통신 관계인 두 통신의 순서쌍이 되고, 이 순서쌍은 IC와 동일하게 정의된다. 그러므로 길이가 1인 ICS는 IC가 되고, IC는 기본 단위의 ICS가 된다.

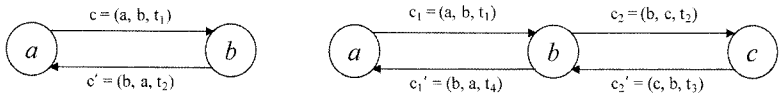
ICS(a, d)의 첫 통신의 발생 시각과 마지막 통신의 발생 시각을 각각 *start-time*(ICS(a, d))와 *end-time*(ICS(a, d))라고 표기하고, *end-time*(ICS(a, d)) - *start-time*(ICS(a, d))는 ICS(a, d)의 응답시간(*response time*)이 된다. 예를 들면, 그림 2(b)에서 *start-time*(ICS(a, d))는 *time*(c_1), *end-time*(ICS(a, d))는 *time*(c_1')이고, ICS(a, d)의 응답시간은 *time*(c_1') - *time*(c_1)이다. ICS는 자신을 구성하는 통신을 순서대로 표기한 통신 순서열 $<c_1, c_2, c_3, c_3', c_2', c_1'>$로 나타낼 수 있을 뿐 아니라, 사용자 사이에 메시지가 전달되는 순서를 표기한 사용자 순서열(*user sequence*)로 나타낼 수 있다. 예를 들어 그림 2(b)의 통신 순서열은 $<c_1, c_2, c_3, c_3', c_2', c_1'>$이고 사용자 순서열은 $a \rightarrow b \rightarrow c \rightarrow d \rightarrow c \rightarrow b \rightarrow a$ 가 된다.

그림 3(a)에는 인버스 페어가 IC인지 판단할 때 사용하는 타임윈도우의 값으로 a가 제시되어 있다. 타임윈도우의 값 a는 다음과 같이 정해진다. 사용자 a가 사용자



(a) IC: $t_2 - t_1 \leq W$ (b) ICS : $t_6 - t_1 \leq W$

그림 2 대화형 통신과 대화형 통신 순서열



(a) IC: $t_2 - t_1 \leq W = \alpha$ (b) ICS: $W = 2\alpha$

그림 3 ICS의 타임윈도우

b에게 시각 t_1에 메시지를 보낸다. 이 때, 메시지를 받은 사용자 b가 사용자 a에게 바로 응답하는 것이 아니라, a의 메시지와 관련된 작업을 수행한 후 시각 t_2에 사용자 a에게 응답 메시지를 보낸다고 가정한다. 예를 들면, 사용자 a가 보낸 메시지의 내용이 "지난 시즌에 매출이 크게 향상된 지점을 찾고, 매출 향상에 기여한 사원들의 정보를 조사해서 보고하시오." 같은 경우이다. 사용자 b가 응답을 하기 위해서 작업을 수행하는데 필요한 시간이 일반적으로 α일을 넘지 않는다고 가정하면, 실제 메시지를 보내고 받는데 소요되는 시간은 α일에 비해 무시할 수 있는 적은 시간이 된다. 그러므로 사용자 a가 사용자 b에게 메시지를 보낸 후, 사용자 b의 응답을 받기까지 필요한 시간은 일반적으로 α일을 초과하지 않을 것이다. 따라서 a를 타임윈도우 W의 값으로 사용하고, 기본 타임윈도우 값(*size of the basic time window*)으로 정한다. 즉, 그림 3(a)에서 a는 InvP(c,c')이 대화형 통신이 되기 위해서 사용자 b가 작업수행을 끝마쳐야 하는 제한 시간이 된다.

그림 3(b)의 통신 순서열에서는 사용자 a가 사용자 b에게 메시지를 보내면, 사용자 b가 자신의 작업만 수행한 후 사용자 a에게 응답하는 것이 아니다. 사용자 b는 사용자 a의 메시지와 관련된 작업을 수행하는 도중에 사용자 c에게 다른 작업을 요청하는 메시지를 보내게 된다. 사용자 c가 사용자 b로부터 받은 메시지와 관련된 작업을 수행한 후, α일 이내에 사용자 b에게 응답을 한다면 $<c_2, c_2'>$는 IC인 동시에 ICS가 된다. 사용자 c로부터 응답을 받고, 자신의 작업을 끝마친 후에 사용자 b는 사용자 a에게 응답 메시지를 보내게 된다. 이 때, $<c_2, c_2'>$는 α일 이내에 끝나서 ICS가 되었지만, $<c_1, c_2, c_2', c_1'>$는 사용자 b의 작업 수행 시간이 추가되는 바람에 α일 이내에 응답되지 않는 경우가 생긴다. 이런 경우에는 사용자 c의 작업을 위한 시간과 사용자 b의 작업을 위한 시간을 고려해서 ICS가 되기 위한 타임윈도우 W를 정한다. 즉, 사용자 b의 작업을 위한 시간 α일을 기본 타임윈도우 α일에 더한 2α일이 ICS가 되기 위한 타임윈도우 W가 된다. 그러므로 사용자 a가 사용자 b에게 메시지를 보낸 시점에서 2α일 이내에 사용자 b의 응답을 받았다면, 통신 순서열 $<c_1, c_2, c_2', c_1'>$는 ICS가 된다.

하지만, 사용자 b의 작업수행과 사용자 c의 작업수행이 병행 될 수 있으므로 α일 이내에 두 작업이 모두 끝날 수도 있다. 이런 경우에는 타임윈도우 W가 확장될 필요가 없다. 즉, $<c_1, c_2, c_2', c_1'>$가 α일 이내에 모두 응답 되었다면, 타임윈도우의 확장을 고려할 필요 없이 $<c_1, c_2, c_2', c_1'>$는 ICS가 된다. 즉, 그림 3(b)의 통신 순서열 $<c_1, c_2, c_2', c_1'>$가 ICS가 되기 위해서는 우선 $<c_2, c_2'>$가 ICS가 되어야 한다. 이 때, $<c_2, c_2'>$가 ICS가 되기 위한 타임윈도우 W의 값은 α이다. 그리고 사용자 a가 b에게 응답 받기까지 소요된 시간 $time(c_1') - time(c_1)$가 a보다 작거나 같은 경우, $<c_1, c_2, c_2', c_1'>$는 ICS가 되고 $<c_1, c_2, c_2', c_1'>$의 타임윈도우 W의 값은 α이다. 반면에 $time(c_1') - time(c_1)$가 α보다 큰 경우, $<c_1, c_2, c_2', c_1'>$의 타임윈도우 W의 값은 2α가 되고, $time(c_1') - time(c_1)$가 W보다 작거나 같은 경우에 $<c_1, c_2, c_2', c_1'>$는 ICS가 된다. 이런 성질에 의해 대화형 통신 순서열을 정의한다.

정의 3. 통신 c와 c'이 서로 역통신 관계이고, 기본 타임윈도우 값 α가 주어졌을 때, ICS(s, d)로 표기되는 대화형 통신 순서열은 다음과 같이 재귀적으로 정의된다:

(1) s=*sender*(c), d=*receiver*(c')이고, $time(c') - time(c) \leq \alpha$이면, ICS($s$, d)::=$<c, c'>$.이 때, ICS(s, d)의 타임윈도우 W는 a로 정해진다.

(2) s=*sender*(c), b=*receiver*(c), *start-time*(ICS(b, d))>*time*(c), *end-time*(ICS(b, d))<*time*(c')이고, $time(c') - time(c) \leq W$이면, ICS($s$, d)::=$<c$, ICS(b, d), $c'>$. 이 때, ICS(s, d)의 타임윈도우 W는 ICS(b, d)의 타임윈도우 W' ($time(c') - time(c) \leq W'$인 경우)이거나 $W' + a$ ($W' \leq time(c') - time(c) \leq W' + a$인 경우)로 정해진다.

일반적으로 통신의 내용을 모두 알 수는 없고, ICS의 정의에 통신의 내용은 고려되지 않는다. 그러므로 IC와 ICS는 통신 순서열 상의 통신들이 서로 연관되어 있을 가능성이 비교적 높다는 것을 의미할 뿐, 실제로 그 통신들의 내용이 밀접하게 연관되어 있음을 보장하지는 않는다.

정의 4. 대화형 통신 순서열 내에서 통신을 주고 받는 사용자의 순서를 해당 대화형 통신 순서열의 사용자 순서열이라고 할 때, 임의의 사용자 순서열 s의 *support*

는 주어진 통신기록(*communication log*) D에 존재하는 모든 ICS 중에 사용자 순서열이 s와 동일한 ICS의 개수를 의미한다. 만일 s의 *support*가 사용자가 정한 하한값 *minsup*보다 크거나 같으면, s는 대화형 통신 순서열 패턴(ICSP: Interactive Communication Sequence Pattern)이 된다.

ICSP는 일반적인 순서열 패턴과 달리 길이가 k인 순서열이 그것을 구성하는 길이가 k-1인 순서열보다 더 많이 나타날 수 있다. 그림 4에서 $c \rightarrow d \rightarrow c$는 $<c_3, c_4>$ 1개가 나타나지만, $b \rightarrow c \rightarrow d \rightarrow c \rightarrow b$는 $<c_2, c_3, c_4, c_5>$, $<c_2, c_3, c_4, c_6>$ 2개가 나타난다. 이것은 ICS(c, d)는 ICS(b, d)를 구성하는 요소지만, ICS(c, d)의 출현 빈도수가 ICS(b, d)보다 적은 경우가 된다. 이 특성에 의해 기존 순서열 패턴 마이닝 알고리즘에서 사용하는 *support* 하한값을 사용한 제거(pruning)를 ICSP를 찾을 때는 적용할 수 없다.

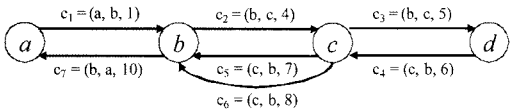


그림 4 ICSP의 특성을 보여주는 예

2.1 문제 정의

통신기록 D, 최대 타임윈도우 W_{max}, 기본 타임윈도우 α, 그리고 *support*의 하한값 *minsup*이 주어졌을 때, 대화형 통신 순서열 패턴 마이닝(*interactive communication sequence pattern mining*)문제의 목적은 통신기록 D에서 모든 대화형 통신 순서열 패턴(ICSP)을 찾는 것이다.

2.2 Naive 방법

다음과 같은 대화형 통신 순서열 패턴을 찾기 위한 직관적인 방법을 Naive 방법이라고 한다. 먼저 통신기록에 존재하는 임의의 통신 (a, b, t_1)에 대해 그 역통신 (b, a, t_n)을 찾는다. 그리고 t_1과 t_n 사이에 발생한 통신 중에 b로 시작하는 통신 (b, x, t_2)와 그 역통신 (x, b, t_m)을 찾는다. 같은 방식으로 t_2와 t_m 사이에 발생한 통신 중에 x로 시작하는 통신과 그 역통신을 찾아나간다. 더 이상 두 통신 사이에 다른 인버스 페어가 발견되지 않을 때까지 찾아 나가면 a에서 시작해서 다시 a까지 돌아오는 ICS형태의 통신 순서열을 찾을 수 있다. 찾아진 순서열이 ICS의 조건을 만족하는 지를 검사하는 방식으로 진행해 나가면, 통신기록에 존재하는 모든 ICS를 찾을 수 있다. 그 중에서 나타나는 빈도수가 하한값 *minsup* 이상인 ICS들이 ICSP가 된다.

3. FDICSP 알고리즘

본 논문에서 제시하는 대화형 통신 순서열 패턴 마이닝 알고리즘인 Fast Discovering Interactive Communication Sequence Pattern(FDICSP)는 통신기록, 최대 타임윈도우, 기본 타임윈도우, 그리고 *support* 하한값을 입력 받아서 통신기록 내의 모든 ICSP를 찾아내기 위해 세 단계의 작업을 수행한다. 첫 번째 단계에서는 통신기록의 모든 인버스 페어를 찾아내서 ICS집합과 ICS는 아니지만 ICS의 부분이 될 가능성이 있는 인버스 페어의 집합 C-ICS집합을 만든다. 그리고 두 번째 단계에서는 ICS로 분류된 인버스 페어들을 조합하여 새로운 ICS를 생성하고, 세 번째 단계에서는 C-ICS의 인버스 페어와 ICS를 조합하여 새로운 ICS를 생성한다. 세 번째 단계가 끝난 후, ICS 집합에서 빈번하게 존재하는 사용자 순서열을 추출하면 FDICSP의 목적인 ICSP를 찾을 수 있다. 3.1, 3.2, 3.3에서 FDICSP의 각 단계에 대해서 자세히 살펴보고, 3.4에서 FDICSP의 시간 복잡도를 분석한다.

3.1 인버스 페어 분류

FDICSP의 첫 번째 단계는 사용자가 입력하는 통신기록 D, 기본 타임윈도우 α, 최대 타임윈도우 W_{max}, 그리고 *support* 하한값 *minsup* 중에서 D, α, 그리고 W_{max}를 사용하고, 다음 단계를 위해 집합 S_{ICS}와 집합 $S_{C\text{-}ICS}$를 생성한다. 집합 S_{ICS}는 ICS를 저장하는 집합이고, 집합 SC-ICS는 ICS가 아니지만 후에 ICS의 부분이 될 수 있는 인버스 페어를 저장한 집합이다.

첫 번째 단계가 시작되면 통신기록 D의 통신 이벤트 중에서 서로 역통신 관계인 통신들을 짝을 지어서 인버스 페어를 생성한다. 생성된 인버스 페어 InvP(c, c')의 응답시간 $time(c')-time(c)$가 α보다 작거나 같은 경우 (c, c')를 S_{ICS}에 추가하고, α보다 큰 경우에는 $S_{C\text{-}ICS}$에 추가한다. 다음은 FDICSP의 첫 번째 단계의 작업 순서이다.

1. 집합 S_{ICS}와 집합 $S_{C\text{-}ICS}$를 생성한다.
2. 발생 시각이 가장 빠른 통신 이벤트 c를 선택한다.
3. 선택된 통신 이벤트 c가 통신기록 D의 마지막 통신 이벤트가 아니면 다음을 수행한다. c가 통신기록 D의 마지막 통신 이벤트라면 4를 수행한다.
   - 통신기록 D를 c의 발생시각 + 1부터 c의 발생시각 + α 사이에 발생한 통신을 탐색하면서 c의 역통신을 찾는다. 역통신 c'이 발견되면, (c, c')을 집합 S_{ICS}에 추가한다.
   - 탐색을 계속한다. c의 발생시각 + W_{max}까지 발생한 통신을 탐색하면서 c의 역통신을 찾는다. 역통신 c'이 발견되면, 인버스 페어 (c, c')을 집합 $S_{C\text{-}ICS}$에 추가한다.
   - c의 다음 통신이 새로운 c로 선택되고, 3을 다시

수행한다.

• 집합 S_{ICS}와 집합 $S_{C\text{-}ICS}$를 반환하고, 종료한다.

통신기록 D에 존재하는 모든 인버스 페어를 집합 S_{ICS}와 집합 $S_{C\text{-}ICS}$에 나누어 저장하면, 인버스 페어 분류 단계가 종료된다.

3.2 기본 타임윈도우 내에서 대화형 통신 순서열 찾기

FDICSP의 두 번째 단계는 사용자가 입력한 기본 타임윈도우 α를 사용하고, 첫 번째 단계에서 생성된 집합 S_{ICS}를 사용한다. 그리고 다음 단계를 위해 집합 S_{ICS}를 반환한다. 두 번째 단계에서 반환하는 집합 S_{ICS}는 응답시간이 기본 타임윈도우 α보다 작거나 같은 모든 ICS를 저장한 집합이다.

이 단계에서 생성해서 사용하는 집합은 $S_{SimpleICS}$, S_{NewICS}, 그리고 S_{Seed}이다. 새로운 ICS는 그림 5처럼 서로 다른 두 개의 ICS가 조합되어 생성된다. 두 번째 단계에서는 조합의 왼쪽에 위치하는 것이 길이가 1인 ICS이고, 오른쪽에 위치하는 것은 길이의 제약이 없는 ICS이다. 첫 번째 패스에서 길이가 1인 두 개의 ICS를 조합하여 길이가 2인 ICS를 생성하고, 두 번째 패스에서는 길이가 1인 다른 ICS와 첫 번째 패스에서 생성된 ICS를 조합하여 길이가 3인 ICS를 생성하는 방식으로 패스에 따라서 점점 길이가 긴 ICS를 생성해 나간다. 이 때, 왼쪽의 길이가 1인 ICS는 첫 번째 단계에서 생성된 ICS집합과 동일한 집합 $S_{SimpleICS}$에 저장된 ICS들을 사용한다. 그리고 각 패스에서 생성된 ICS는 집합 S_{NewICS}에 저장되는데, 그 ICS들은 다음 패스에서 집합 S_{Seed}에 저장되어 조합의 오른쪽에 놓이게 된다.

두 번째 단계에서 두 ICS가 조합되어 새로운 ICS를 생성하는 조건은 조합의 왼쪽에 위치하는 ICS $\langle c, c'\rangle$와 오른쪽에 위치하는 ICS(x, y)가 $start\text{-}time(ICS(x, y)) > time(c)$와 $end\text{-}time(ICS(x, y)) < time(c')$를 만족하는 것이다. S_{ICS}의 ICS들은 모두 응답시간이 α 이내이기 때문에 위의 조건만 만족하면 두 ICS를 조합해서 새로운 ICS를 만들 수 있다. 예를 들면, $t_6 - t_1 < \alpha$, $t_5 - t_2 < \alpha$, $t_4 - t_3 < \alpha$이고, $t_1 < t_2 < t_3 < t_4 < t_5 < t_6$이면, ICS$(b, d)$와 ICS$(a, d)$가 그림 5처럼 생성된다. 다음은 FDICSP의 두 번째 단계의 작업 순서이다.

1. 집합 $S_{SimpleICS}$와 집합 S_{NewICS}를 생성하고, 집합 S_{ICS}의 모든 원소를 두 집합에 복사한다.
2. 집합 S_{NewICS}가 공집합이면 5를 수행한다. 공집합이 아니면 다음을 수행한다.
   - 집합 S_{Seed}를 새로 생성하고, 집합 S_{NewICS}의 모든 원소를 집합 S_{Seed}에 복사한다.
   - 집합 S_{NewICS}를 공집합으로 초기화한다.
   - 집합 $S_{SimpleICS}$의 ICS 중 하나인 ics를 선택한다.
3. ics와 조합되어 새로운 ICS를 생성할 수 있는 ICS를 집합 S_{Seed}에서 모두 찾는다. 새로 생성한 ICS는 집합 S_{NewICS}에 추가한다.
4. 집합 $S_{SimpleICS}$의 ICS 중 아직 선택되지 않은 것을 새로운 ics로 선택하고, 3을 다시 수행한다. 만일 집합 $S_{SimpleICS}$의 모든 ICS가 선택되어서 새로 선택할 ICS가 없다면, S_{NewICS}의 모든 ICS를 S_{ICS}에 추가하고, 2를 수행한다.
5. 집합 S_{ICS}를 반환하고, 종료한다.

응답시간이 기본 타임윈도우 α보다 작거나 같은 모든 ICS를 찾아서 집합 S_{ICS}에 저장하면, 기본 타임윈도우 내에서 대화형 통신 순서열 찾기 단계가 종료된다.

3.3 타임윈도우를 확장하며 대화형통신 순서열 찾기

FDICSP의 세 번째 단계는 사용자가 입력한 기본 타임윈도우 α와 *support* 하한값 *minsup*을 사용하고, 첫 번째 단계에서 생성된 집합 $S_{C\text{-}ICS}$와 두 번째 단계에서 ICS가 추가된 집합 S_{ICS}를 사용한다. 그리고 최종 결과로써 ICSP를 반환한다.

세 번째 단계는 두 번째 단계와 유사하다. 차이점은 그림 5처럼 새로운 ICS를 생성할 때, 조합의 왼쪽에 위치하는 것이 ICS가 아니라 C-ICS의 인버스 페어라는

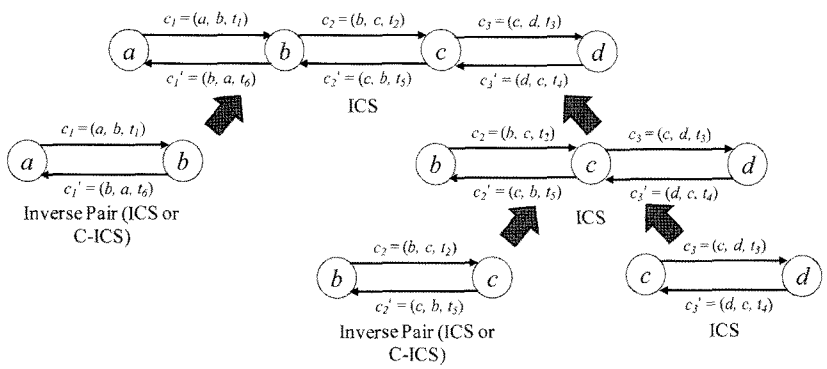


그림 5 새로운 ICS를 생성하는 방법

것이다. 그리고 조합의 오른쪽에 위치하는 ICS는 이전의 두 단계에서 생성한 모든 ICS가 된다.

세 번째 단계에서 두 ICS가 조합되어 새로운 ICS를 생성하는 첫 번째 조건은 조합의 왼쪽에 위치하는 인버스 페어 $<c, c'>$와 오른쪽에 위치하는 ICS(x, y)가 *start-time*(ICS(x, y)) > *time*(c)와 *end-time*(ICS(x, y)) < *time*(c')를 만족하는 것이다. 두 번째 조건은 *time*(c') - *time*(c)가 ICS(x, y)의 타임윈도우 + α 보다 작거나 같아야 한다. 두 조건이 모두 만족할 경우에만 $<c$, ICS(x, y), $c'>$는 새로운 ICS가 된다. 이 때, *time*(c') - *time*(c) ≤ ICS(x, y)의 W(타임윈도우)인 경우에는 새로운 ICS의 W는 ICS(x, y)의 W와 동일한 값을 갖게 되고, ICS(x, y)의 W ≤ *time*(c') - *time*(c) ≤ α + ICS(x, y)의 W인 경우에는 새로운 ICS의 W는 α + ICS(x, y)의 W가 된다. 예를 들면, 그림 5에서 $t_6 - t_1 < 3\alpha$, $t_5 - t_2 < 2\alpha$, $t_4 - t_3 < \alpha$이고, $t_1 < t_2 < t_3 < t_4 < t_5 < t_6$ 이면, ICS(b, d)는 W가 2α인 ICS가 되고, ICS(a, d)가 W가 3α인 ICS가 된다. 하지만, $t_6 - t_1 > 2\alpha$, $t_5 - t_2 < \alpha$, $t_4 - t_3 < \alpha$이고, $t_1 < t_2 < t_3 < t_4 < t_5 < t_6$ 이면, ICS(b, d)는 W가 α인 ICS가 되고, $<c_1, c_2, c_3, c_3', c_2', c_1'>$는 *time*($c_1'$) - *time*($c_1$)가 α (ICS(b, d)의 타임윈도우) + α 보다 크기 때문에 ICS가 되지 못한다. 다음은 FDICSP의 세 번째 단계의 작업 순서이고, 그림 6은 이 단계에 대한 의사코드(pseudo code)이다.

1. 집합 S_{NewICS}를 생성하고, 집합 S_{ICS}의 모든 원소를 집합 S_{NewICS}에 복사한다.
2. 집합 S_{NewICS}가 공집합이면 5를 수행한다. 공집합이 아니면 다음을 수행한다.
   - 집합 S_{Seed}를 생성하고, 집합 S_{NewICS}의 모든 원소를 집합 S_{Seed}에 복사한다.
   - 집합 S_{NewICS}를 공집합으로 초기화한다.
   - 집합 $S_{C\text{-}ICS}$의 인버스 페어 중 하나인 c-ics를 선택한다.
3. c-ics와 조합되어 새로운 ICS를 생성할 수 있는 ICS를 집합 S_{Seed}에서 모두 찾는다. 새로 생성한 ICS는 집합 S_{NewICS}에 추가한다.
4. 집합 $S_{C\text{-}ICS}$의 인버스 페어 중 아직 선택되지 않은 것을 새로운 c-ics로 선택하고, 3을 다시 수행한다. 만일 집합 $S_{C\text{-}ICS}$의 모든 인버스 페어가 선택되어서 새로 선택할 인버스 페어가 없다면, 집합 S_{NewICS}의 모든 ICS를 S_{ICS}에 추가하고 2를 수행한다.
5. 집합 S_{ICS}에서 ICSP를 추출하여 반환하고, 종료한다.

S_{ICS}의 ICS들의 사용자 순서열(*user sequence*) 중에서 *support*가 *minsup* 이상인 사용자 순서열을 모두 추출하면, 그 순서열들이 ICSP가 된다. ICSP를 반환하고, 타임윈도우를 확장하며 대화형 통신 순서열 찾기 단계를 종료한다.

3.4 시간 복잡도 분석

첫 번째 단계의 탐색비용은 통신기록의 통신 이벤트 수 n과 최대 타임윈도우 W_{max}에 의해서 결정된다. 첫 번째 단계에서는 통신기록의 각 통신 c에 대해 c의 발생시각 + 1부터 c의 발생시각 + W_{max}까지 진행하면서 역통신을 찾으므로, 각 통신에 대해 최대 W_{max} 만큼의 탐색이 수행된다. 그러므로, 첫 번째 단계의 탐색비용은 W_{max} * n = O(n)이 된다.

Algorithm Step 3 of FDICSP [Finding ICS with time window expansion]

Input α: basic time window
minsup: minimum frequency
S_{ICS}: a set of interactive communications
$S_{C\text{-}ICS}$: a set of inverse pairs that are not interactive communication
Output S_{ICS}: a set of interactive communications
Procedure
01: $S_{NewICS} \leftarrow S_{ICS}$
02: **while** (S_{NewICS} is not empty)
03: $S_{Seed} \leftarrow S_{NewICS}$
04: $S_{NewICS} \leftarrow \{\}$
05: **foreach** communication (c, c') in $S_{C\text{-}ICS}$
06: **if** (c, c') and an ICS(a, d) in S_{Seed} can create a new ICS by using time window expansion
07: $S_{NewICS} \leftarrow S_{NewICS} \cup \{ <c, ICS(a, d), c'> \}$
08: **end if**
09: **end foreach**
10: **foreach** ICS(s, d) in S_{NewICS}
11: $S_{ICS} \leftarrow S_{ICS} \cup \{ ICS(s, d) \}$
12: **end foreach**
13: **end while**

그림 6 타임윈도우 확장하며 대화형 통신 순서열 찾기

두 번째 단계의 탐색비용은 통신기록의 통신 이벤트 수 n과 기본 타임윈도우 값 α에 의해서 결정된다. 통신기록의 각 통신 c에 대해 최대 α개의 대화형 통신이 존재할 수 있다. 그러므로 첫 번째 단계에서 생성되는 S_{ICS}의 크기는 $\alpha * n$ 보다 작다. 두 번째 단계의 각 패스(pass)에서는 집합 $S_{SimpleICS}$의 각 ICS에 대해서 집합 S_{Seed}의 모든 ICS를 한번씩 탐색하므로 탐색비용은 $N(S_{SimpleICS}) * N(S_{Seed})$가 된다. 두 집합의 크기의 최대값은 $N(S_{ICS})$의 최대값인 $\alpha * n$과 같으므로, 두 번째 단계의 각 패스의 최대 탐색비용은 $(\alpha * n) * (\alpha * n) = O(n^2)$가 된다. 서로 역통신인 통신 이벤트가 하나의 ICS를 이루기 때문에, 통신 이벤트의 수가 n개일 때 생성될 수 있는 가장 긴 ICS의 길이는 n/2이다. 길이가 n/2인 ICS가 포함된 통신기록이 입력 되었다면, 그 ICS를 찾기 위해 n/2 - 1 번의 패스가 수행되어야 하므로 두 번째 단계의 최대 탐색비용은 $O(n^3)$이 된다. 하지만, 일반적으로 ICS의 길이는 길지 않으므로 두 번째 단계 전체의 최대 탐색비용은 $O(n^2)$이 된다.

세 번째 단계의 시간 복잡도는 두 번째 단계와 유사하다. 집합 $S_{SimpleICS}$ 대신 집합 $S_{C\text{-}ICS}$가 사용되므로, 각 패스의 최대 탐색비용은 $N(S_{SimpleICS}) * N(S_{Seed})$ 대신 $N(S_{C\text{-}ICS}) * N(S_{Seed})$가 된다. 그러므로 세 번째 단계의 각 패스의 최대 탐색비용은 $O(n^2)$가 된다. 길이가 n/2인 ICS가 포함된 통신기록이 입력된다면, 세 번째 단계의 최대 탐색비용은 $O(n^3)$가 되지만, 일반적으로 ICS의 길이는 길지 않으므로 세 번째 단계 전체의 최대 탐색비용은 $O(n^2)$이다.

최악의 경우에는 두 번째 단계와 세 번째 단계의 탐색비용이 $O(n^3)$이 될 수 있기 때문에 FDICSP의 탐색비용은 $O(n^3)$이 된다. 하지만, 일반적인 경우에는 첫 번째 단계의 탐색비용은 O(n), 두 번째 단계는 $O(n^2)$, 세 번째 단계는 $O(n^2)$이므로 FDICSP의 탐색비용은 $O(n^2)$이 된다.

4. 실 험

실험을 통해 다른 알고리즘과 성능비교를 하여 FDICSP의 성능을 분석하였다. 성능비교의 대상으로 2.2절에서 설명한 Naive 방법과 기존방법의 개념을 적용하여 만든 *Exhaustive(Apriori-like)*, *Exhaustive(Pattern Growth-like)* 방법을 사용하였다.

Exhaustive(Apriori-like) 방법과 *Exhaustive(Pattern Growth-like)* 방법은 엄밀하게 *Apriori*, 혹은 *Pattern Growth*라고 할 수 없지만, 최대한 각 방법의 개념을 적용하여 만든 방법들이다. 기존방법에서는 순서열의 각 구성요소가 독립적으로 존재하지만, ICSP에서는 인버스 페어라는 순서쌍으로 존재하는 것을 반영하여야 했다. 그리고 *minsup*을 1로 지정해서 수행해야만 했는데, 그 이유는 길이가 k인 ICSP가 그것을 구성하는 길이가 k-1인 ICSP보다 더 많이 나타날 수 있는 ICS의 특성 때문이다.

실험은 Intel Core2 Duo 2.67GHz 프로세서와 2GB 메인메모리를 탑재한 시스템에서 수행되었고, 알고리즘은 C#으로 구현되었다. 성능의 측정 기준은 주어진 통신기록에서 모든 대화형 통신 순서열 패턴을 찾아내는데 수행되는 시간과 메모리 사용량을 사용하였다. 입력되는 통신기록으로는 랜덤 함수를 사용하여 생성된 통신 이벤트 들을 사용하였다.

4.1 통신기록의 크기에 따른 성능 비교

그림 7은 α = 250, W_{max} = 3000, *minsup* = 1으로 고정되었을 때, 통신기록의 크기에 따른 수행 시간 비교이다.

Exhaustive(Apriori-like) 방법은 ICS가 될 가능성 여부를 고려하지 않고 모든 순서열을 찾기 때문에 통신의 수가 많으면, 작업을 수행하는 도중에 불필요하게 생성되는 순서열이 증가한다. 그러므로 그림 7처럼 성능저하를 보이게 된다. *Exhaustive(Pattern Growth-like)* 방법은 프로젝트된 데이타베이스를 만들고, 각 데이타베이스 안의 통신끼리만 ICS가 될 수 있는지 여부를 검사한다. 그 이유는 서로 다른 프로젝트된 데이타베이스에 속한 통신들은 조합되어 ICS를 생성할 수 없기 때문이다. 그 결과 통신의 수가 증가에 따라 증가하는 불필요한 검사 작업이 적기 때문에 성능저하가 크지 않았다.

Naive 방법은 통신의 수에 따라 ICS가 아니지만 ICS인지 여부를 체크해야 하는 대상이 많아지므로 성능의 저하가 나타난다. 반면에, FDICSP는 통신기록의 증가한 부분에서 ICS가 될 수 있는 통신에 대해서 탐색작업을 수행하고, 증가한 통신 중에 ICS가 되는 통신의 수가 비교적 적기 때문에 큰 성능저하가 나타나지 않는다.

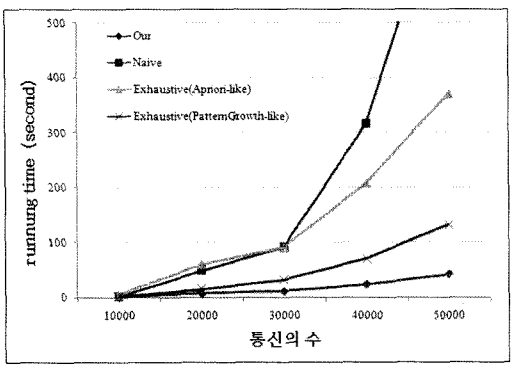


그림 7 통신기록 크기에 따른 수행 시간 비교(α = 250, W_{max} = 3000, minsup = 1)

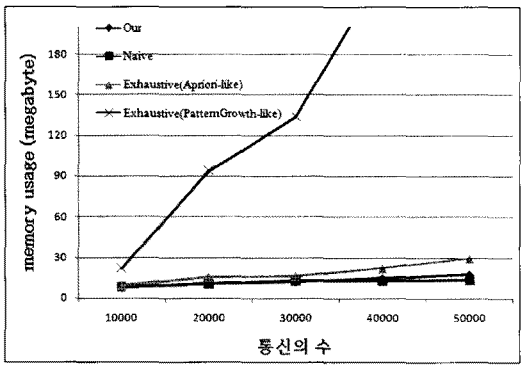


그림 8 통신기록 크기에 따른 메모리 사용량 비교(α = 250, W_{max} = 3000, minsup = 1)

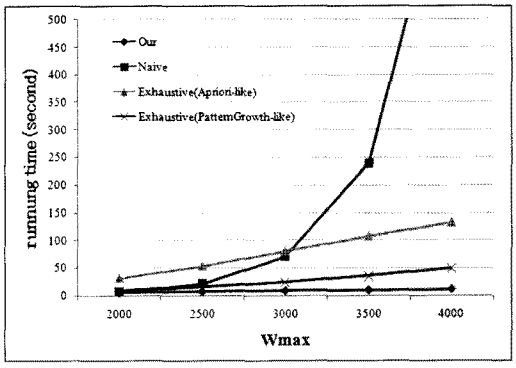


그림 9 W_{max}의 크기에 따른 수행 시간 비교(# of dataset = 25000, α = 250, minsup = 1)

그림 8은 α = 250, W_{max} = 3000, *minsup* = 1으로 고정되었을 때, 통신기록의 크기에 따른 메모리 사용량 비교이다.

Naive 방법은 처리를 빠르게 하기 위한 특별한 데이타 구조를 가지고 있지 않기 때문에 메모리 사용량은 제일 적었다. *Exhaustive(Apriori-like)* 방법은 통신의 수가 적을 경우에는 소량의 메모리만 필요했지만, 통신의 수가 많아짐에 따라서 각 단계에서 생성되는 수많은 순서열들을 저장하기 위한 메모리가 필요하여 점차 소비량이 증가했다.

수행 속도에서는 위의 두 방법보다 뛰어났던 *Exhaustive(Pattern Growth-like)*방법은 프로젝트된 데이타베이스를 만드는 과정에서 엄청난 양의 메모리를 사용했고, 통신의 수의 증가에 따라 엄청난 소비량의 증가가 나타났다. 반면에 FDICSP는 Naive 방법보다는 많은 양의 메모리가 필요했지만, 두 *Exhaustive* 방법보다는 적은 메모리를 사용했다. 통신의 수가 증가하는 경우에도 메모리 요구량이 크게 변하지 않기 때문에 *Exhaustive(PatternGrowth-like)*방법보다 훨씬 적은 양의 메모리를 사용하면서도 좋은 성능을 보였다.

4.2 W_{max}의 크기에 따른 성능 비교

통신의 수, α, *minsup*이 고정되었을 때, W_{max}의 크기에 따른 메모리 사용량을 측정하였을 때, 세 방법 모두 실험을 수행한 통신의 수에 따라 사용량이 변할 뿐, W_{max}의 변화에는 영향을 받지 않았다.

그림 9는 통신의 수는 25,000개, α = 250, *minsup* = 1로 고정되었을 때, W_{max}의 크기에 따른 수행시간 비교이다.

Naive 방법에서는 W_{max}가 증가하면서 탐색 대상이 되는 통신 c의 역통신 c'의 개수가 증가할 뿐만 아니라 그 사이에 존재하는 인버스 페어의 수도 증가한다. 더 많은 인버스 페어를 고려하고, 각 인버스 페어 사이에 존재하는 인버스 페어를 찾아 들어가는 깊이가 깊어지게 되기 때문에 비교작업의 횟수는 급격히 증가한다. 그리고 *Exhaustive(Apriori-like)*방법은 W_{max}가 증가하면, 통신의 수가 많을 때와 같이 불필요하게 찾게 되는 순서열의 수가 증가하게 되고, 그것이 성능의 저하로 이어진다.

반면에 *Exhaustive(PatternGrowth-like)*방법은 가능성이 있는 통신들만 ICS가 될 수 있는지 여부를 검사하는 프로젝트된 데이타베이스를 사용하기 때문에 W_{max}의 증가에 큰 영향을 받지 않았다. 그리고 FDICSP도 W_{max}의 증가에 따라 탐색 및 비교 횟수가 증가하기는 하지만, ICS가 될 가능성이 있는 통신에 대해서만 초점을 맞추기 때문에 성능의 저하가 거의 없다.

4.3 α의 크기에 따른 성능 비교

통신의 수, W_{max}, *minsup*이 고정되었을 때, α의 크기에 따른 성능 비교에서는 세 방법 모두 실험을 수행한 통신의 수와 W_{max}에 따라 수행시간과 메모리 사용량이 변할 뿐, α의 변화에 의해서는 변화를 보이지 않았다.

5. 결 론

본 논문에서는 대화형 통신 순서열 이라는 순서열을 정의하였고, 대화형 통신 순서열 패턴 마이닝 문제를 소개하였다. 대화형 통신 순서열은 길이가 k인 순서열이 그것을 구성하는 길이가 k-1인 순서열보다 더 많이 나타날 수 있는 특성을 갖기 때문에, 기존의 순서열 패턴 마이닝을 사용할 경우 제대로 된 결과를 얻지 못하거나, 도중에 순서열을 제거하는 작업을 제외하여 큰 성능저하를 감수해야 한다. 이에 본 논문에서는 대화형 통신 순서열의 특성을 고려하여 대화형 통신 순서열 패턴 마이닝 문제를 해결하기 위한 FDICSP 알고리즘을 제안하였고, 실험을 통해서 FDICSP는 통신기록의 크기가 큰 경우와 최대 타임윈도우가 크게 설정된 경우에도 성

능이 저하되지 않으면서 ICSP를 찾아내는 것을 확인했다. 특히 큰 데이타 셋(data set)을 사용해도 성능저하가 거의 없었다는 점에서 효율적으로ICSP를 찾아내고 있음을 확인할 수 있었다.

참 고 문 헌

[1] DIRECTIVE 2006/24/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 15 March 2006 on the retention of data generated or processed in connection with the provision of publicly available electronic communications services or of public communications networks and amending Directive 2002/58/EC.

[2] ETSI TS 101 331 - Telecommunications security; Lawful Interception(LI); Requirements of Law Enforcement Agencies.

[3] ETSI TS 101 671 - Telecommunications security; Lawful Interception(LI); Handover interface for the lawful interception of telecommunications traffic.

[4] Philip A. Branch. Lawful Interception of the Internet. Centre for Advanced Internet Architectures. Technical Report 030606A.

[5] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering frequent episodes in sequences," Proc. of the Int'l Conference on Knowledge Discovery in Databases and Data Mining(KDD-95), pp. 210-215, 1995.

[6] R. Agrawal and R. Srikant, "Mining Sequential Patterns: Generalizations and Performance Improvements," Advances in Database Technology - EDBT'96, 5th International Conference on Extending Database Technology, pp. 3-17, 1996.

[7] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," Proc. of the 20th VLDB Conference, pp. 487-499, 1994.

[8] M. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences," Machine Learning 40, pp. 31-60, 2001.

[9] S. Aseervatham, A. Osmani and E. Viennet, "Bitspade: A Lattice-Based Sequential Pattern Mining Algorithm Using Bitmap Representation," Proc. Sixth Int'l Conf. Data Mining(ICDM), pp. 792-797, 2006.

[10] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth," Proc. of the 17th International Conference on Data Engineering (ICDE'01), pp. 215-224, 2001.

[11] J. Pei, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U Dayal, and M. Hsu, "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach," IEEE Transactions on Knowledgeand Data Engineering 16, pp. 1424-1440, 2004.

[12] C. Yu and Y. Chen, "Mining Sequential Patterns from Multidimensional Sequence Data," IEEE Transactions on Knowledge and Data Engineering, v.17, n.1, pp. 136-140, 2005.

[13] C. Raissi and M. Plantevit, "Mining Multidimensional Sequential Patterns over Data Streams," LNCS 5182, pp. 263-272, 2008.

[14] M. Plantevit, A. Laurent, M. Teisseire, "HYPE: mining hierarchical sequential patterns," Proc.of the 9th ACM international workshop on Data warehousing and OLAP(DOLAP), pp. 19-26, 2006.

함 덕 민

정보과학회논문지 : 데이타베이스
제 36 권 제 1 호 참조

송 지 환

2002년 2월 서강대학교 컴퓨터공학과 학사. 2004년 8월 한국과학기술원 전산학과 석사. 2004년 9월~현재 한국과학기술원 전산학과 박사과정. 관심분야는 데이타베이스 시스템, 그리드 기반 분산 시스템, 데이타 마이닝 등

김 명 호

정보과학회논문지 : 데이타베이스
제 36 권 제 1 호 참조