High Utility Episode Mining Made Practical and Fast

Guangming Guo[1,2], Lei Zhang[3], Qi Liu[1],
Enhong Chen[1], Feida Zhu[2], and Chu Guan[1]

[1] School of Computer Science and Technology
University of Science and Technology of China, Hefei 230027, China
[2] School of Information Systems, Singapore Management University
[3] School of Computer Science and Technology, Anhui University
guogg@mail.ustc.edu.cn

Abstract. This paper focuses on the problem of mining high utility episodes from complex event sequences. Episode mining, one of the fundamental problems of sequential pattern mining, has been continuously drawing attention over the past decade. Meanwhile, there is also tremendous interest in the problem of high utility mining. Recently, the problem of high utility episode mining comes into view from the interface of these two research areas. Although prior work [11] has proposed algorithm UP-Span to tackle this problem, their method suffers from several performance drawbacks. To that end, firstly, we explicitly interpret the high utility episode mining problem as a complete traversal of the lexicographic prefix tree. Secondly, under the framework of lexicographic prefix tree, we examine the original UP-Span algorithm and present several improvements on it. In addition, we propose several clever strategies from a practical perspective and obtain much tighter utility upper bounds of a given node. Based on these optimizations, an efficient algorithm named TSpan is presented for fast high utility episode mining using tighter upper bounds, which reduces huge search space over the prefix tree. Extensive experiments on both synthetic and real-life datasets demonstrate that TSpan outperforms the state-of-the-art in terms of both search space and running time significantly.

1 Introduction

Sequential pattern mining [6], one of the most important data mining problems, has been continuously drawing attention from researchers. And when the sequential data becomes event sequence, the task of *frequent episodes mining* (FEM) [7] is introduced. FEM reveals a lot of useful information hidden in the event sequence with a wide range of applications [2,4,7,8,10]. However, the discovered frequent episode is still too simple and primitive. In some cases, FEM may lose some important information. It only takes the presence and absence of events into account and neglects the semantic information of different events. However, in reality, events in the same sequence can be significantly different from each other. For example, different web pages/items are of different importance/cost/profit for the decision makers (both producers and consumers). If we only apply FEM to event sequences, some truly interesting episodes, such as high-profit ones, may be filtered out due to their low frequency.

X. Luo, J.X. Yu, and Z. Li (Eds.): ADMA 2014, LNAI 8933, pp. 71–84, 2014.

To tackle the above challenge, the concept of utility is introduced as an alternative measure aside from frequency [3,12]. In *high utility mining*, unlike the traditional frequency based pattern mining, the *downward closure property* does not keep. In other words, when appending a new item/event to an itemset/episode one by one, the frequency of an itemset/episode decreases monotonously or remains unchanged, but the utility varies irregularly. As a result, previous optimization methods in frequent pattern mining become invalid in high utility mining. Without efficient pruning strategies, high utility mining becomes prohibitive due to the curse of dimensionality. Fortunately, exploiting *transaction/sequence weighted utility* (*TWU/SWU*) [5,13], we can first generate candidates efficiently using *TWU/SWU*'s downward closure property, and then identify high utility ones from far smaller number of candidates.

However, previous high utility mining works mainly focus on the transactional databases, seldom consider other types of databases, like sequence database. As far as we know, Wu et al. [11] presented the first attempt to solve the problem of *high utility episode mining*[1] in complex event sequence. But the proposed algorithm UP-Span suffers from low efficiency in both running time and memory consumption. More importantly, the proposed utility upper bound named *Episode Weighted Utility* (*EWU*) is only a loose and basic utility bound for episodes. In practice, we may need to check a large number of candidates when using *EWU*, which becomes a bottleneck for high utility episode mining in large databases. To that end, in this paper we present several improvements over UP-Span, and an efficient algorithm named TSpan, which has two much tighter upper bounds than *EWU* for high utility episode mining.

Contribution. In this paper, we first explicitly tackle high utility episode mining under the framework of a traversal of lexicographic prefix tree. Under this framework, we discuss how to implement the UP-Span algorithm in a much more efficient manner, which can save a lot of search space and running time. Moreover, leveraging the lexicographic prefix tree, we propose algorithm named TSpan (**T**ighter upper bound when **Span**ning prefixes), which consists of two tighter upper bounds for operations related to spanning episodes. These upper bounds are able to maintain the preferable downward closure property, which effectively reduces search space over the tree. Last but not least, we conduct experiments with both synthetic and real-life datasets and clearly show that TSpan can effectively reduce both search space and running time.

2 Preliminaries and Lexicographic Prefix Tree

Fig. 1 shows a simple complex event sequence, where each event in the sequence is associated with an event type and occurrence time. It is called complex event sequence because at each occurrence time there can be events occurring simultaneously and the frequency of events at a given time is different, as shown in Fig. 1. Following [11], events occur at the same time point are called simultaneous events SE. For high utility episode mining, each event is associate with an external utility value and Table 1 shows an example utility table for the example event sequence in Fig. 1.

[1] Following [11], when talking about the word episode, we specifically refer to serial episode. Other type of episodes are good candidates for future research.

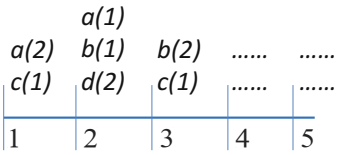


Fig. 1. A simple complex event sequence

Table 1. External utilities for events in Fig. 1

Event	a	b	c	d
Utility	1	1	1	1.5

Introduced in [7], the number of minimal occurrences is a popular measure for frequent episode mining. Formal definition of minimal occurrence is as follows: an occurrence of an episode $\alpha, [T_s, T_e]$, is minimal iff α does not occur at any subinterval $[T_s^{'}, T_e^{'}] \subset [T_s, T_e]$. For simplicity, here we denote a minimal occurrence of episode α as $mo(\alpha)$. The set of all minimal occurrences of α is denoted as $moSet(\alpha)$. For instance, in Fig. 1, the time interval [1,2] is one of the minimal occurrences of episode $\langle ab \rangle$ and $moSet(\langle a, b \rangle) = \{[1, 2], [2, 3]\}$.

Based on minimal occurrence, we can define the utility of an episode. Given an episode $\alpha = \langle SE_1, SE_2, ..., SE_k \rangle$, where each simultaneous events SE_i is associated with a time point T_i, the utility of episode α w.r.t. minimal occurrence $mo(\alpha) = [T_s, T_e]$ is $u(\alpha, mo(\alpha)) = \sum_{i=1}^{k} u(SE_i, T_i)$. For example, $u(\langle ab \rangle, [1, 2]) = u(a, T_1) + u(b, T_2) = 2 + 1 = 3$. As one episode may have multiple minimal occurrences, the utility of an episode can be naturally defined as the sum of episode's utility w.r.t. each minimal occurrence, i.e., $u(\alpha) = \sum_{mo(\alpha) \in moSet(\alpha)} u(\alpha, mo(\alpha))/u(CES)$[2]. For episode $\langle ab \rangle$ in Fig. 1, $u(\langle ab \rangle) = (u(\langle ab \rangle, [1, 2]) + u(\langle ab \rangle, [2, 3]))/11 = (3 + 3)/11 = 0.55$.

In real world cases, the events of an episode usually happen together within a reasonable time period. Following [11], we name maximum time duration of an episode as MTD. For any minimal occurrence $mo(\alpha) = [T_s, T_e]$ of α, it must satisfy the condition that $T_e - T_s + 1 \leq MTD$. With MTD, we can then formally introduce the concept *EWU* presented in [11] as follows.

Definition 1 (*EWU*(α)). *Given episode* $\alpha = \langle (SE_1), (SE_2), ..., (SE_n) \rangle$ *and* $mo(\alpha) = [T_s, T_e]$ *is one of its minimal occurrence, EWU (Episode*[3] *Weighted Utility) value of* α *w.r.t.* $mo(\alpha)$*, i.e.,* $EWU(\alpha, mo(\alpha)) = \sum_{i=1}^{n} u(SE_i, T_i) + \sum_{i=e}^{s+MTD-1} u(CES_i, T_i)$. *Then given* $moSet(\alpha) = \{mo(\alpha)_1, mo(\alpha)_2, ..., mo(\alpha)_k\}$, $EWU(\alpha) = \sum_{i=1}^{k} EWU(\alpha, mo(\alpha)_i) / u(CES)$.

[2] For simplicity of comparison, episode's utility is defined a ratio between its utility value and complex event sequence's total utility value.

[3] Under the current context, the term "episode" specifically denotes the largest episode when expanding α given a user specified MTD.

Similar to *TWU/SWU*, *EWU* serves as upper bound of an episode's utility and maintains the favorable downward closure property. Using this upper bound, the candidate-generation-and-test mechanism can then be effectively used in high utility episode mining. In this way, the computing cost of high utility episode mining becomes acceptable.

Different from [11], we view high utility episode mining from complex event sequences as a complete traversal over the lexicographic prefix tree. Under this framework, a more clear understanding of the process can be obtained, and further development of optimization becomes easier. Before formally introducing lexicographic prefix tree, we would like to first define two operations over the prefix tree, i.e., I-Concatenation and S-Concatenation, which are closely related to this concept.

Definition 2 (I-Concatenation and S-Concatenation). *Assume that we have an* l*-episode*[4] α*, appending an event to the end of* α *will lead to an expanded episode, which is an* $(l+1)$*-episode. We call this operation as* concatenation. *Specifically, when the time duration of* α *does not change, we denote this operation as* I-Concatenation. *However, when the time duration of* α *is increased by 1, we denote this operation as* S-Concatenation.

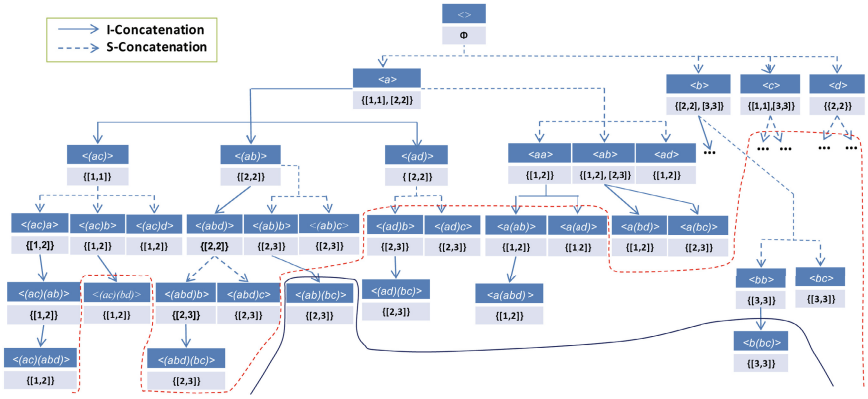


Fig. 2. Lexicographic prefix tree for event sequence in Fig. 1 (MTD=2, min_util=0.7)

Definition 3 (Lexicographic prefix tree). *In lexicographic prefix tree, a) the root node*[5] *of the prefix tree is empty, b) all the child nodes are generated from the* I-Concatenation *or* S-Concatenation *of a parent node, c) all the child nodes are listed in an incremental and lexicographic order.*

Fig. 2 shows an example lexicographic prefix tree for the example event sequence $\langle(ac)(abd)(bc)\rangle$ in Fig. 1. In Fig. 2, we also present the minimal occurrences associated with each episode. As it can be seen, even for such short an event sequence with MTD=2 and min_util=0.7, large number of nodes need checking if there is no pruning strategy employed. In Fig. 2, the black solid line shows the search space boundary over the

[4] An l-episode means that the episode's length is l.

[5] Without ambiguity, the terms episode and node will be used interchangeably in the following.

prefix tree when only using $EWU(\alpha)$ as the upper bound estimation of episode's utility for pruning. In the following, we will also take Fig. 2 as example for explaining our improvements over UP-Span and proposed pruning strategies. For easy reference, we summarize the notations defined above in Table 2.

Table 2. Summary of notations described above

CES	The total complex event sequence
SE_k	Simultaneous events at time T_k
$mo(\alpha)$	The minimal occurrence of episode α
$moSet(\alpha)$	The set of all minimal occurrences of episode α
$u(\alpha)$	The utility value of α
MTD	Maximum time duration of episodes
$EWU(\alpha)$	Episode Weighted Utility of α
I-Concatenation	Concatenation operation that increases episode length by 1, but episode's time duration size keeps unchanged
S-Concatenation	Concatenation operation that increases episode length by 1, and also increases episode's time duration size by 1

3 Efficient High Utility Episode Mining

In this section, we will first discuss the improvements over the original UP-Span algorithm we have made. Secondly, we will present our proposed pruning strategies, which are more tighter and efficient than *EWU*. Our strategies effectively improve the algorithm's efficiency by further reducing the search space. In the example lexicographic prefix tree (Fig. 2) for event sequence $\langle(ac)(abd)(bc)\rangle$, the search space boundary using our pruning strategies is shown by the dashed cut line (red). Compared to *EWU*, this example clearly shows that our proposed upper bound based pruning strategies are much more effective and efficient.

3.1 Efficient Implementation of the UP-Span Algorithm

The algorithm UP-Span (*high Utility ePisode mining by Spanning prefixes*) presented in [11], adopts idea similar to that of [13,9], all following the same *prefix-growth* (also known as *pattern-growth*) paradigm. When spanning prefixes in the complex event sequences, there are two kinds of candidate events, i.e., the simultaneous events and serial events, to extend with. Thus, we need to define two different procedures for spanning episode prefixes in high utility episode mining. The UP-Span algorithm uses the term *miningSimultHUE* and *miningSerialHUE* to denote these two procedures. Readers can refer to [11] for the details of the UP-Span algorithm. However, we'd like to adopt *I-Concatenation* and *S-Concatenation* [13] to denote these two different procedures, since these two terms explicitly indicating that these two operations are on the lexicographic prefix tree. The details of our implementation of *I-Concatenation* and *S-Concatenation* are shown in Algorithm 1 and 2. The procedure named ***Prefix-Growth*** is omitted since it is simply a successive call of these two operations.

Algorithm 1. I-Concatenation

Input : (1) α: episode (2) $moSet(\alpha)$: all minimal occurrences of α (3) MTD: maximum time duration (4) $min\_utility$: minimum utility threshold

Output: The set of high utility simultaneous episodes w.r.t prefix α

for *each* $mo(\alpha) = [T_s, T_e] \in moSet(\alpha)$ **do**
  $IES = \{e \mid$ event e occurs at T_e *and* e is after the last event in $\alpha\}$;
  for *each event* $e \in IES$ **do**
    $\beta = I\text{-}concatenate(\alpha, e)$;
    $mo(\beta) = mo(\alpha)$;
    $betaSet = betaSet \cup \beta$;
    $moSet(\beta) = moSet(\beta) \cup mo(\beta)$;
for *each* $\beta \in betaSet$ **do**
  if *EWU*$(\beta) \geq min\_utility$ **then**
    if $u(\beta) \geq min\_utility$ **then**
      $HUE\_Set = HUE\_Set \cup \beta$;
    Prefix-Growth$(\beta, moSet(\beta), MTD, min\_utility)$;

Algorithm 2. S-Concatenation

Input : (1) α: episode (2) $moSet(\alpha)$: all minimal occurrences of α (3) MTD: maximum time duration (4) $min\_utility$: minimum utility threshold

Output: The set of high utility simultaneous episodes w.r.t prefix α

for *each* $mo(\alpha) = [T_s, T_e] \in moSet(\alpha)$ **do**
  for *each time point* t *in* $[T_e + 1, T_s + MTD + 1]$ **do**
    $SES_t = \{e \mid$ event e occurs at $t\}$;
    for *each event* $e \in SES_t$ **do**
      $\beta = S\text{-}concatenate(\alpha, e)$;
      $mo(\beta) = [T_s, t]$;
      $M = \{mo \mid mo \in moSet(\beta)\ and\ mo \subseteq mo(\beta)\}$;
      if $M = \emptyset$ **then**
        $N = \{mo \mid mo \in moSet(\beta)\ and\ mo(\beta) \subset mo\}$;
        if $N \neq \emptyset$ **then**
          $moSet(\beta) = moSet(\beta) - N$;
          $moSet(\beta) = moSet(\beta) \cup mo(\beta)$;
        else
          $betaSet = betaSet \cup \beta$;
          $moSet(\beta) = moSet(\beta) \cup mo(\beta)$;
for *each* $\beta \in betaSet$ **do**
  if *EWU*$(\beta) \geq min\_utility$ **then**
    if $u(\beta) \geq min\_utility$ **then**
      $HUE\_Set = HUE\_Set \cup \beta$;
    Prefix-Growth$(\beta, moSet(\beta), MTD, min\_utility)$;

Both *I-Concatenation* and *S-Concatenation* follow the same framework: firstly, generate all the valid candidates $betaSet$ and their corresponding minimal occurrence set $moSet(\beta)$ by means of prefix growth from episode α (the first **for** loop); then each candidate is checked to see if its *EWU* value is above the threshold $min\_utility$ to decide whether or not to recursively call the procedure ***prefix-growth*** (the second **for** loop). If the candidate β's exact utility is above $min\_utility$, it will be added to *HUE_Set* (Line 11 and 19 respectively). After this recursive procedure *prefix-growth* is executed on all the 1-episodes in lexicographic order, it exactly finishes the complete search on the lexicographic prefix tree and all the high utility episodes will be collected into the set *HUE_Set*. It is worth noting that the sub-procedure *I-Concatenate/S-Concatenate* performs simultaneous/serial concatenation operation on α and $e \in IES/SES$ to form candidate episode β (Line 4 and 5 respectively), and the sub-procedure *EWU* and $u(\alpha)$ computes the *EWU* and *utility* value of an episode respectively.

Discussion. Although the main procedure of the above presented algorithms is the same as the UP-Span algorithm, we make a couple of improvements compared to the original UP-Span algorithm.

First of all, we conduct the search process explicitly under the framework of lexicographical prefix tree. Before the search process begins, the simultaneous events at each time point are sorted in the lexicographic order as it is required in lexicographical tree. Consequently, we have the following nice property: Using the I-Concatenation procedure described in Algorithm 1, 1) all the candidate simultaneous episodes $betaSet$ with prefix α will be generated, and 2) each $\beta \in betaSet$ has exactly the complete and correct minimal occurrence set $moSet(\beta)$[6].

Secondly, when spanning prefixes using *S-Concatenation* (Algorithm 2), we do not simply first collect all the serial candidate episode sets $betaSet$ and their corresponding minimal occurrence set $moSet(\beta)$, and then filter out the illegal minimal occurrences of each β. Instead, whenever adding a new minimal occurrence of β, we check if there exists a minimal occurrence mo in $moSet(\beta)$ such that $mo \subseteq mo(\beta)$ (this set of mo is denoted as M in Line 7) *or* $mo \supset mo(\beta)$ (this set of mo is denoted as N in Line 9). If M is not $\emptyset$, then $mo(\beta)$ is not a valid minimal occurrence for episode β, and no further steps are taken. On the other hand, if M is $\emptyset$ (Line 8), and N is not $\emptyset$, the set N will be deleted from the $moSet(\beta)$ (N is not minimal occurrence because of the new $mo(\beta)$); otherwise, $mo(\beta)$ will be directly added to $moSet(\beta)$ (Line 10-15).

Furthermore, as we can see, different from the UP-Span algorithm, the projected database is not utilized in these algorithms, which is a common trick in sequential pattern mining to facilitate the prefix-growth process. It is because that not only the episode α itself, but also its minimal occurrence set $moSet(\alpha)$ are stored during the mining process. Together with the original event sequence and MTD, $moSet(\alpha)$ already provides a sparse representation of projected database w.r.t. α, which can be used to compute the *EWU* value and *utility* of α efficiently.

The final optimization of our implementation is that we propose to compute the exact utility of candidate episodes incrementally since the prefix-growth paradigm generates patterns recursively. As every minimal occurrence $mo(\beta)$ of the candidate episode β

[6] The proof is omitted due to space limit.

will be generated, $u(\beta, mo(\beta))$ can be directly obtained from the sum of the corresponding $mo(\alpha)$'s utility and the newly appended event e's utility. Thereafter, the utility of β can be obtained from the sum of $u(\beta, mo(\beta))$. In this way, we can avoid the frequent scans of the original database to compute the value of $u(\beta)$.

3.2 Efficient Pruning Strategies

Although the pruning strategies proposed using *EWU* is effective as demonstrated in [11], we argue here that the performance of this algorithm can be further improved. Our proposed upper bounds are able to further prune the search space over the lexicographic prefix tree and reduce the running time of algorithm significantly.

Table 3. Notations used in the following

$E_{last}(\alpha)$	The last event of α
$IES_i(\alpha)$	Event Set for *I-Concatenation* w.r.t. $mo_i(\alpha)$
$SES_i(\alpha)$	Event Set for *S-Concatenation* w.r.t. $mo_i(\alpha)$
$IES(\alpha)$	Event Set for *I-Concatenation* w.r.t. α
$SES(\alpha)$	Event Set for *S-Concatenation* w.r.t. α

Improved *EWU* for I-Concatenation As described in **Definition** 1, the computation of *EWU(α)* w.r.t. one minimal occurrence $mo(\alpha) = [T_s, T_e]$ is the sum of two parts, episode α's own utility w.r.t. $mo(\alpha)$, and the partial utility of CES starting from time point T_e to $T_s + MTD - 1$. Taking all the minimal occurrences of α into account, we can then get *EWU*(α) by summing all the *EWU*$(\alpha, mo_i(\alpha))$. However, the estimated utility upper bound is still very loose. In fact, the first and second part have overlaps with each other during computation. That is, if $\alpha =< SE_s, SE_{s+1}, ..., SE_e >$, the utility of event set SE_e is counted twice during computation.

Under the framework of lexicographic prefix tree, events in each simultaneous event set are sorted in the lexicographic order. Hopefully, the repetitive calculation of utility of the events $e \in SE_e$ can be avoided. Specifically, we propose to compute an upper bound in this way: the first part keeps unchanged; the second part becomes the sum of utility of events after E_{last} in CES_e, i.e. $u(IES(\alpha))$, and utility of events in subsequent CES_i after CES_e, i.e. $u(SES(\alpha))$. As there often exist events in the set SE_e, this method can be seen as an improved upper bound of $u(\alpha)$ before I-Concatenation. We denote this new upper bound as *IEIC* (Improved *EWU* for *I-Concatenation*).

$$IEIC(\alpha, mo_i(\alpha)) = \sum_{j=1}^{k} u(SE_j, T_j) + u(IES_i(\alpha)) + u(SES_i(\alpha)), \qquad (1)$$

where $\alpha =< (SE_1), (SE_2), ..., < SE_j >, ..., (SE_k) >$, $mo_i(\alpha) = [T_{si}, T_{ei}]$, and the meaning of $IES_i(\alpha)$ and $SES_i(\alpha)$ is illustrated in Table 3.

Taking all the minimal occurrences $mo(\alpha)$ of episode α into consideration, we can then obtain the value of $IEIC(\alpha)$:

$$IEIC(\alpha) = (u(\alpha) + u(IES(\alpha)) + u(SES(\alpha)))/u(CES). \qquad (2)$$

This newly proposed upper bound $IEIC(\alpha)$, can be directly applied to replace the function of *EWU* used in Algorithm 1, and 2. In this way, we get a more efficient upper bound for pruning the search space.

Improved *EWU* for S-Concatenation. When it comes to *S-Concatenation* in the ***prefix-growth*** procedure, an improved upper bound of $u(\alpha)$ can be obtained similar to *IEIC*. Any *S-concatenation* episode β of α w.r.t. $mo(\alpha) = [T_s, T_e]$ only considers the event $e \in \{CES_i \mid e+1 \leq i \leq s+MTD-1\}$, i.e., $SES(\alpha)$. Therefore, there is no need to take the utility of event set $IES(\alpha)$ into account when estimating the upper bound of $u(\alpha)$ for *S-Concatenation*. Similar to *IEIC*, we denote this new upper bound as *IESC* (Improved estimation of *EWU* for S-Concatenation).

$$IESC(\alpha) = (u(\alpha) + u(SES(\alpha)))/u(CES). \tag{3}$$

This efficient estimation of utility upper bound can only be applied before the sub-procedure of *S-Concatenation*. Thus, the ***prefix-growth*** procedure for high utility episode mining can be improved by employing this upper bound. We call this updated ***prefix-growth*** procedure as ***TSpan***, which is illustrated as follows.

Algorithm 3. TSpan

Input : (1) α: episode (2) $moSet(\alpha)$: all minimal occurrences of α (3) MTD: maximum time duration (4) $min\_utility$: minimum utility threshold
Output: The set of high utility episodes with α as the prefix

1 ***I-Concatenation***$(\alpha, moSet(\alpha), MTD, min\_utility)$;
2 **if** *IESC*$(\alpha) \geq min\_utility$ **then**
3 ⌊ ***S-Concatenation***$(\alpha, moSet(\alpha), MTD, min\_utility)$;

Discussion. Replacing function *EWU(α)* with *IEIC(α)* and procedure *Prefix-Growth* with TSpan (Algorithm 3) in Algorithm 1, 2, we finally obtain our improved algorithm TSpan over UP-Span. As TSpan adopts strategies including tighter upper bound estimation of $u(\alpha)$ and tighter upper bound estimation for *S-Concatenation*, much more search space is pruned compared to UP-Span. The efficiency of TSpan can be shown clearly in the example lexicographic prefix tree in Fig. 2. The black solid line shows the search space boundary of the UP-Span algorithm, while the red dashed line shows the search space boundary of TSpan.

To be specific, the total utility of $u(\langle(ac)(abd)(bc)\rangle) = 11$ under the above settings. Therefore, the utility of a given node $\alpha, i.e., u(\alpha)$ must be greater than $7.7(11 * 0.7)$ to be recognized as a high utility episode. Using UP-Span, only nodes $\langle(ab)(bc)\rangle$ and $\langle(b)(bc)\rangle$ are pruned, while TSpan is much more efficient at reducing search space in this example. For instance, the child nodes of $\langle(ad)\rangle$ are not pruned by UP-Span since *EWU*$(\langle\langle(ad)\rangle\rangle) = (u(\langle\langle(ad)\rangle\rangle) + u(\langle\langle(abd)(bc)\rangle\rangle))/11 = 1.1$, greater than 0.7. But *IEIC*$(\langle\langle(ad)\rangle\rangle) = (u(\langle\langle(ad)\rangle\rangle) + u(\langle\langle(bc)\rangle\rangle))/11 = 7/11 < 0.7$, thus the child nodes of $\langle(ad)\rangle$ is not searched using our proposed algorithm. Before performing *S-Concatenation* operation on node $\langle(b)\rangle$, the original upper bound *EWU*$(\langle\langle(b)\rangle\rangle)$ is sure to be larger than 0.7 due to two minimal occurrences of $\langle(b)\rangle$. But *IESC*$(\langle\langle(b)\rangle\rangle) =$

$(u(\langle(b)\rangle, [2,2]) + u(\langle(bc)\rangle) + u(\langle(b)\rangle, [3,3]))/11 = 6/11 < 0.7$. As a result, all the serial concatenation episodes of $\langle(b)\rangle$ are pruned from the search space. Similar results can also be demonstrated for other pruned nodes.

4 Performance Evaluation

In this section, we will conduct experiments on various algorithms to evaluate the performance of our proposed strategies.The main characteristics of datasets used in experiments are presented in Table 4. The synthetic dataset is generated with the IBM synthetic data generator for transactional database [1]. The main parameters to generate these dataset include T: the average size of transactions; I: the average size of maximal potential pattern; D: the total number of transactions; N: the number of distinct items. Aside from synthetic dataset, three real-life datasets are also employed during evaluation. The first one is Foodmart2000 database[7], a well known example database for business intelligence. The second one is Retail dataset, obtained from the FIMI dataset repository[8]. The last one is Chainstore dataset, a large dataset downloaded from NU-MineBench repository[9].

Table 4. Statistical information of different datasets

DataSet	#Trans	#Items	Avg. Length
T20I12N1KD10K	10,000	1,000	20
FoodMart2000	5,581	1,559	15.6
Retail	88,162	8,600	11.2
ChainStoreSmall	10,000	46,086	14.3

It should be noted that both the synthetic and real-life datasets can be seen as transactional databases. But they can also be viewed as a single long complex event sequence by considering each item as an event and each transaction as a simultaneous event set at a time point. In the experiments, we only use the first 10k transactions of dataset Retail and ChainStore as this size is enough for comparison and bigger size of the dataset will cost too much time to run. Only the FoodMart2000 and Chainstore dataset have unit profits (i.e., external utility) and product sales (i.e., internal utility). Therefore, we generate unit profits and product sales as follows: the unit profits of each item are generated using a log-normal distribution ranging from 1 to 1000 and sales of each items are generated randomly between 1 and 5.

To evaluate the proposed algorithm for high utility episodes mining, we compare TSpan with three baseline algorithms, that is, the original UP-Span algorithm (UP-Span), improved algorithm with strategy IEIC (IEIC Only), improved algorithm with strategy IESC (IESC Only) and our proposed algorithm TSpan with

[7] `http://msdn.microsoft.com/en-us/library/aa217032(v=sql.80).aspx`

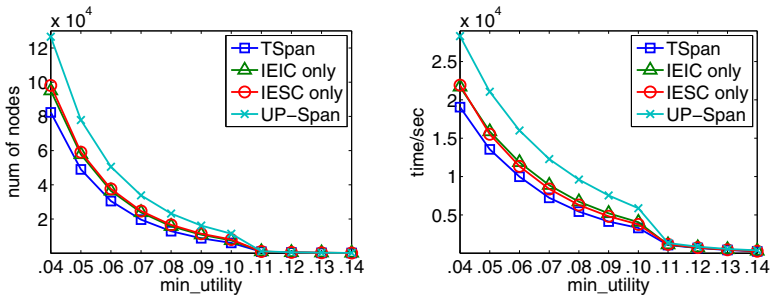
[8] `http://fimi.ua.ac.be/data/`

[9] `http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html`

both strategies (TSpan). As the optimization over the implementation of operations *I-Concatenation* and *S-Concatenation* in Section 3.1 is in very fine granularity and is sure to save running time and memory consumption, we focus on the performance comparison between TSpan and UP-Span. It is worth noting that both the baselines and TSpan here adopt the implementation improvement in Section 3.1.

4.1 Evaluation on Synthetic Dataset

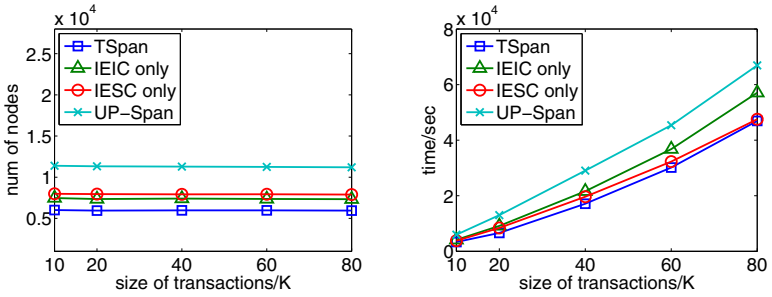
As shown in Fig. 3, the tendency between search space and running time are very consistent. That is, the more the searched nodes, the longer the running time. Both the proposed strategies IEIC and IESC take effects, reducing the running time and search space as much as 25% when $min\_utility = 0.05$. And the combination of both proposed pruning strategies leads to further improved efficiency. As it can be seen, the difference between the four algorithms' performance becomes larger when the $min\_utility$ decreases, and we can forecast that the gap between algorithms will be larger as $min\_utility$ continues to decrease. Since experiments are conducted under the setting that MTD = 8 and utilities of events follow the lognormal distribution, high utility episodes can only be generated when $min\_utility$ is very small. All these facts lead to the conclusion that the proposed strategies are practically useful.



(a) Number of Nodes (b) Running Time

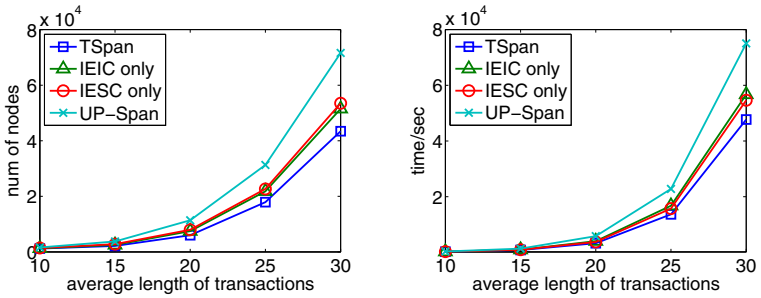
Fig. 3. Evaluation under varied $min\_utility$ on dataset T20I12N1KD10K (MTD=8)

We also tested the scalability of the four algorithms when the size of the transactions (i.e., the size of the complex event sequence) increases. The results are reported in Fig. 4. As we generate the dataset using the same algorithm and all other settings are the same, the number of searched nodes keeps the same. But the running time of these algorithms grows linearly, indicating that the algorithms' scalability is very good. In Fig. 5, we vary the average length of transactions, i.e., the average length of simultaneous event set, to compare the performance difference between the algorithms. TSpan grows smoothly in both number of searched nodes and running time, while the baselines grow much faster, demonstrating that TSpan is more efficient.



(a) Number of Nodes (b) Running Time

Fig. 4. Evaluation under varied #Trans on dataset T20I12N1KDxK ($min\_utility$=0.1, MTD=8)

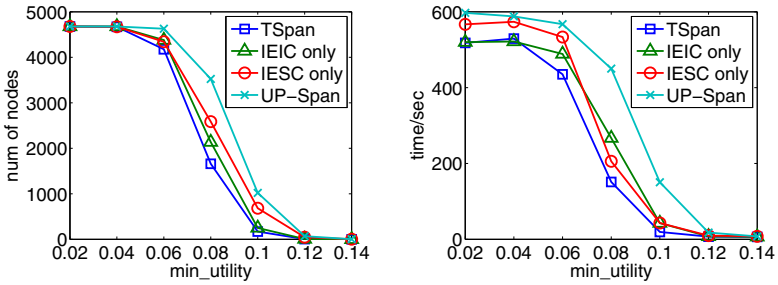


(a) Number of Nodes (b) Running Time

Fig. 5. Evaluation under varied Avg.Length on dataset TxI12N1KD10K ($min\_utility$=0.1, MTD=8)

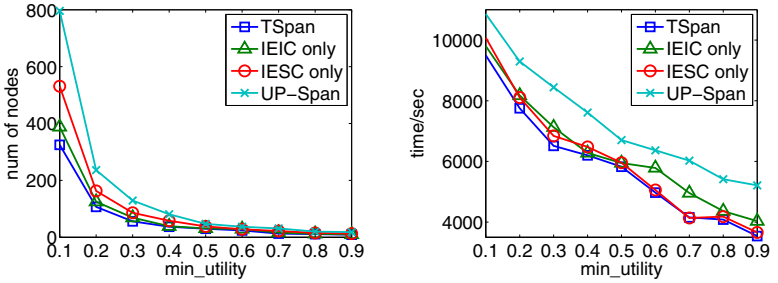
4.2 Evaluation on Real-life Dataset

Fig. 6 shows the performance comparison on real-life dataset FoodMart2000. As this dataset is quite small and sparse, the execution time of the four algorithms is very short. However, we can still clearly observe that the proposed algorithms outperform the baseline by a large margin. For example, when $min\_utility = 0.08$, TSpan finishes in only one fifth of the time needed by UP-Span. For dataset Retail, due to its inherent characteristics and our settings on external utilities, the running time of these algorithms becomes too long to observe when $MTD > 4$. So we only report the performance comparison when $MTD = 4$, which is still capable of demonstrating the performance advantage of our proposed strategies.We can predict from the curve's trend of Fig. 7 (a) that the gap between the TSpan and the baselines will become even larger when $min\_utility$ is smaller than 0.1. In Fig. 7 (b), we can conclude that TSpan is faster than the UP-Span algorithm by more than 2000 seconds in most cases. Fig. 8 shows that TSpan is always the winner compared with others, similar to the other two real-life datasets. And strategy IEIC and IESC can compensate each other well to further improve the algorithm's performance.



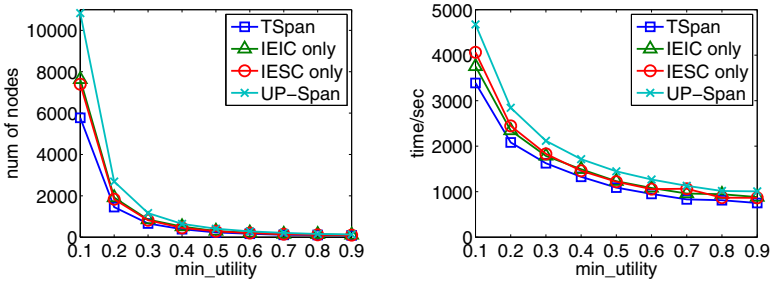
(a) Number of Nodes (b) Running Time

Fig. 6. Evaluation under varied minimum utility thresholds on dataset FoodMart2000 (MTD=8)



(a) Number of Nodes (b) Running Time

Fig. 7. Evaluation under varied minimum utility thresholds on dataset Retail (MTD=4)



(a) Number of Nodes (b) Running Time

Fig. 8. Evaluation under varied minimum utility thresholds on dataset ChainStoreSmall (MTD=6)

5 Conclusion and Future Work

In this paper, we focus on proposing practical and fast high utility episode mining algorithms. The most important thing we did is that we tackle the problem under the framework of a complete traversal of the lexicographic prefix tree. Using this framework, we first presented four efficient improvements over the original implementation

of UP-Span. Secondly, we proposed two novel strategies named $IEIC$ and $IESC$ to obtain tighter upper bounds of a given node's utility, which bring about the efficient algorithm TSpan. Finally, we demonstrated the effectiveness of these strategies systematically on both synthetic and real life datasets. Experimental results show that the proposed strategies can improve the performance significantly by reducing the number of searched nodes and the running time. We believe that these proposed strategies can be incorporated with other similar high utility mining tasks, and more effective strategies can be proposed using lexicographic prefix tree in the future.

Acknowledgement. This work was supported in part by the National High Technology Research and Development Program of China under Grant No. 2014AA015203, the Anhui Provincial Natural Science Foundation under Grant No. 1408085QF110 and the Fundamental Research Funds for the Central Universities of China under Grant No. WK0110000042. This work was also partially supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office, Media Development Authority (MDA). Lei Zhang is supported by the Academic and Technology Leader Imported Project of Anhui University (NO. J10117700050).

References

1. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In: VLDB, pp. 487–499 (1994)
2. Ahmed, C.F., Tanbeer, S.K., Jeong, B.S.: Mining high utility web access sequences in dynamic web log data. In: SNPD, pp. 76–81 (2010)
3. Chan, R., Yang, Q., Shen, Y.-D.: Mining high utility itemsets. In: ICDM, pp. 19–26 (2003)
4. Lee, W., Stolfo, S.J., Mok, K.W.: A data mining framework for building intrusion detection models. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 120–132 (1999)
5. Liu, Y., Liao, W.K., Choudhary, A.N.: A fast high utility itemsets mining algorithm. In: Workshop on Utility-Based Data Mining (2005)
6. Mabroukeh, N.R., Ezeife, C.I.: A taxonomy of sequential pattern mining algorithms. ACM Comput. Surv. 43(1), 3 (2010)
7. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data Min. Knowl. Discov. 1(3), 259–289 (1997)
8. Ng, A., Fu, A.W.-C.: Mining frequent episodes for relating financial events and stock trends. In: Whang, K.-Y., Jeon, J., Shim, K., Srivastava, J. (eds.) PAKDD 2003. LNCS, vol. 2637, pp. 27–39. Springer, Heidelberg (2003)
9. Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: Prefixspan: Mining sequential patterns by prefix-projected growth. In: ICDE, pp. 215–224 (2001)
10. Saxena, K., Shukla, R.: Significant interval and frequent pattern discovery in web log data. CoRR, abs/1002.1185 (2010)
11. Wu, C.-W., Lin, Y.-F., Tseng, V.S., Yu, P.S.: Mining high utility episodes in complex event sequences. In: KDD (2013)
12. Yao, H., Hamilton, H.J., Butz, C.J.: A foundational approach to mining itemset utilities from databases. In: The 4th SIAM International Conference on Data Mining, pp. 482–486 (2004)
13. Yin, J., Zheng, Z., Cao, L.: Uspan: an efficient algorithm for mining high utility sequential patterns. In: KDD, pp. 660–668 (2012)