SCIENCE CHINA
Information Sciences

• LETTER •

October 2025, Vol. 68, Iss. 10, 209101:1–209101:2
https://doi.org/10.1007/s11432-024-4496-9

OIP-Miner: one-off incremental sequential pattern mining with forgetting factor

Youxi WU[1], Shuang TIAN[1], Yan LI[2], Jing LIU[1*], Jinyan LI[3],
Wenjian WANG[4] & Xindong WU[5]

[1]*School of Artificial Intelligence, Hebei University of Technology, Tianjin 300400, China*
[2]*School of Economics and Management, Hebei University of Technology, Tianjin 300400, China*
[3]*Computer Science and Control Engineering, Shenzhen Institute of Advanced Technology,*
Chinese Academy of Sciences, Shenzhen 518055, China
[4]*School of Computer and Information Technology, Shanxi University, Taiyuan 237016, China*
[5]*Key Laboratory of Knowledge Engineering with Big Data of Ministry of Education,*
Hefei University of Technology, Hefei 230009, China

Received 28 June 2024/Revised 8 March 2025/Accepted 6 June 2025/Published online 12 September 2025

Citation Wu Y X, Tian S, Li Y, et al. OIP-Miner: one-off incremental sequential pattern mining with forgetting factor. Sci China Inf Sci, 2025, 68(10): 209101, https://doi.org/10.1007/s11432-024-4496-9

When a database is incremented, sequential patterns also need to be dynamically maintained [1]. Incremental sequential pattern mining (SPM) avoids the need for brute-force mining across all sequences [2]. Moreover, in some time-sensitive applications, more recent data are generally believed to be more valuable. Therefore, the forgetting mechanism [3] (or decaying factor [4]) can be introduced. Classical incremental SPM methods disregard intra-sequence pattern repetition frequency as a potential indicator of user behavior significance [5], and repetitive SPM methods focus on static data. To overcome these shortages, this paper addresses one-off incremental sequential pattern (OIP) mining with a forgetting factor. OIP mining employs a one-off occurrence counting method for support calculation and introduces a forgetting factor to progressively reduce the impact of old data. To tackle the problem of OIP mining, we propose an OIP-Miner algorithm. To efficiently calculate the supports, OIP-Miner constructs a monomial search dictionary based on an inverted dictionary. To reduce the number of candidate patterns, we propose four pruning strategies based on the pattern join strategy. To improve the efficiency of incremental pattern mining, OIP-Miner mines only incremental data based on a global pattern dictionary.

Problem definition.

Definition 1. The support of pattern $\mathbf{p}$ in sequence $\mathbf{s}$ is the number of one-off occurrences, and is represented by $\mathrm{sup}(\mathbf{p},\mathbf{s})$. The support of pattern $\mathbf{p}$ within one batch is $\mathrm{sup}(\mathbf{p},B)=\sum_{j=1}^{b}\mathrm{sup}(\mathbf{p},\mathbf{s}_j)$. The support of pattern $\mathbf{p}$ in an incremental sequence database D with a forgetting factor is the sum of the product of the support in each batch and the weight of that batch based on the forgetting factor, i.e., $f\mathrm{sup}(\mathbf{p},D)=\sum_{t=1}^{k}(\mathrm{sup}(\mathbf{p},B_t)\times\eta^{k-t})$.

Definition 2. In an incremental sequence database D, if $\mathrm{sup}(\mathbf{p},B)/|B|\geqslant\beta$ $(0<\beta<1)$, then pattern $\mathbf{p}$ is a frequent one-off pattern in the added batch B, where $|B|$ represents the number of itemsets in B. Similarly, if $f\mathrm{sup}(\mathbf{p},D)/|D|\geqslant\beta$, then pattern $\mathbf{p}$ is a frequent OIP. Our goal is to discover all frequent OIPs from the incremental sequence database.

Proposed algorithm. OIP-Miner consists of four parts: data preprocessing, support calculation, candidate pattern generation, and incremental pattern mining. The framework of OIP-Miner is shown in Figure 1.

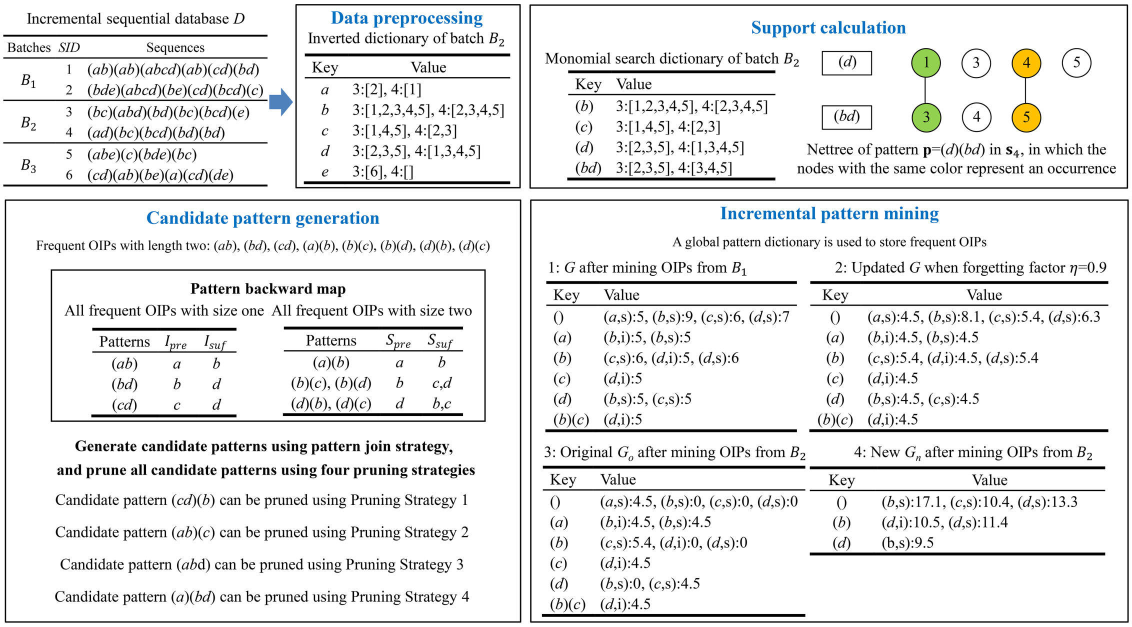
Data preprocessing. We create an inverted dictionary for each item. The dictionary has a key and its value. The key stores the item, and the value is also a dictionary, where its key is an ID of sequence SID, and its value is a list storing the occurrence positions in the sequence.

Support calculation. Based on the inverted dictionary for each item, we create a monomial search dictionary, which has two components: a key and its value. The key stores the frequent OIPs with size one, and the value is also a dictionary, whose key is an ID of sequence SID and whose value is a list that stores the positions of each frequent OIP with size one in the sequence. We create m level nodes of the Nettree using the monomial search dictionary, and employ a depth-first search method to iteratively find the root-leaf paths to calculate the support for the pattern.

Candidate pattern generation. We adopt the pattern join strategy to generate candidate patterns. Suppose we have two frequent OIPs $\mathbf{p}$ and $\mathbf{q}$ with length m. If $\mathrm{suf}(\mathbf{p})=\mathrm{pre}(\mathbf{q})$, then we generate a candidate superpattern $\mathbf{r}$ with length $m+1$, denoted as $\mathbf{r}=\mathbf{p}\oplus\mathbf{q}$.

To further reduce the number of candidate patterns, we propose a pattern backward map structure, which is constructed from frequent OIPs with length two. Its size can

[*] Corresponding author (email: liujhebut@163.com)



Incremental sequential database D

Batches	*SID*	Sequences
B_1	1	(*ab*)(*ab*)(*abcd*)(*ab*)(*cd*)(*bd*)
	2	(*bde*)(*abcd*)(*be*)(*cd*)(*bcd*)(*c*)
B_2	3	(*bc*)(*abd*)(*bd*)(*bc*)(*bcd*)(*e*)
	4	(*ad*)(*bc*)(*bcd*)(*bd*)(*bd*)
B_3	5	(*abe*)(*c*)(*bde*)(*bc*)
	6	(*cd*)(*ab*)(*be*)(*a*)(*cd*)(*de*)

Data preprocessing

Inverted dictionary of batch B_2

Key	Value
a	3:[2], 4:[1]
b	3:[1,2,3,4,5], 4:[2,3,4,5]
c	3:[1,4,5], 4:[2,3]
d	3:[2,3,5], 4:[1,3,4,5]
e	3:[6], 4:[]

Support calculation

Monomial search dictionary of batch B_2

Key	Value
(*b*)	3:[1,2,3,4,5], 4:[2,3,4,5]
(*c*)	3:[1,4,5], 4:[2,3]
(*d*)	3:[2,3,5], 4:[1,3,4,5]
(*bd*)	3:[2,3,5], 4:[3,4,5]

Nettree of pattern $\mathbf{p}$=(*d*)(*bd*) in $\mathbf{s}_4$, in which the nodes with the same color represent an occurrence

Candidate pattern generation

Frequent OIPs with length two: (*ab*), (*bd*), (*cd*), (*a*)(*b*), (*b*)(*c*), (*b*)(*d*), (*d*)(*b*), (*d*)(*c*)

Pattern backward map

All frequent OIPs with size one

Patterns	I_{pre}	I_{suf}
(*ab*)	*a*	*b*
(*bd*)	*b*	*d*
(*cd*)	*c*	*d*

All frequent OIPs with size two

Patterns	S_{pre}	S_{suf}
(*a*)(*b*)	*a*	*b*
(*b*)(*c*), (*b*)(*d*)	*b*	*c*,*d*
(*d*)(*b*), (*d*)(*c*)	*d*	*b*,*c*

Generate candidate patterns using pattern join strategy, and prune all candidate patterns using four pruning strategies

Candidate pattern (*cd*)(*b*) can be pruned using Pruning Strategy 1

Candidate pattern (*ab*)(*c*) can be pruned using Pruning Strategy 2

Candidate pattern (*ab*d) can be pruned using Pruning Strategy 3

Candidate pattern (*a*)(*bd*) can be pruned using Pruning Strategy 4

Incremental pattern mining

A global pattern dictionary is used to store frequent OIPs

1: G after mining OIPs from B_1

Key	Value
()	(*a*,s):5, (*b*,s):9, (*c*,s):6, (*d*,s):7
(*a*)	(*b*,i):5, (*b*,s):5
(*b*)	(*c*,s):6, (*d*,i):5, (*d*,s):6
(*c*)	(*d*,i):5
(*d*)	(*b*,s):5, (*c*,s):5
(*b*)(*c*)	(*d*,i):5

2: Updated G when forgetting factor η=0.9

Key	Value
()	(*a*,s):4.5, (*b*,s):8.1, (*c*,s):5.4, (*d*,s):6.3
(*a*)	(*b*,i):4.5, (*b*,s):4.5
(*b*)	(*c*,s):5.4, (*d*,i):4.5, (*d*,s):5.4
(*c*)	(*d*,i):4.5
(*d*)	(*b*,s):4.5, (*c*,s):4.5
(*b*)(*c*)	(*d*,i):4.5

3: Original G_o after mining OIPs from B_2

Key	Value
()	(*a*,s):4.5, (*b*,s):0, (*c*,s):0, (*d*,s):0
(*a*)	(*b*,i):4.5, (*b*,s):4.5
(*b*)	(*c*,s):5.4, (*d*,i):0, (*d*,s):0
(*c*)	(*d*,i):4.5
(*d*)	(*b*,s):0, (*c*,s):4.5
(*b*)(*c*)	(*d*,i):4.5

4: New G_n after mining OIPs from B_2

Key	Value
()	(*b*,s):17.1, (*c*,s):10.4, (*d*,s):13.3
(*b*)	(*d*,i):10.5, (*d*,s):11.4
(*d*)	(b,s):9.5

Figure 1 (Color online) Framework of OIP-Miner.

be one or two. If it is one, then we use $I_{\rm pre}$ and $I_{\rm suf}$ to store the prefix and suffix of each frequent OIP, respectively. If $I_{\rm pre}$ of pattern $\mathbf{p}$ is x, then $I_{\rm pre}[x]$ is $I_{\rm suf}$ of pattern $\mathbf{p}$. Similarly, if it is two, then we use $S_{\rm pre}$ and $S_{\rm suf}$ to store the prefix and suffix of each frequent OIP, respectively. If $S_{\rm pre}$ of pattern $\mathbf{p}$ is x, then $S_{\rm pre}[x]$ is $S_{\rm suf}$ of pattern $\mathbf{p}$. Based on the pattern backward map, we propose four pruning strategies.

Pruning strategy 1 (pruning the first items in *S*-join). If $\mathrm{len}(q_n) = 1$ and $i_{11} \notin S_{\rm pre}$, then pattern $\mathbf{r}$ cannot be a frequent OIP and can be pruned.

Pruning strategy 2 (pruning the first and last items in *S*-join). If $\mathrm{len}(q_n) = 1$, $i_{11} \in S_{\rm pre}$, and $e_{n1} \notin S_{\rm pre}[i_{11}]$, then pattern $\mathbf{r}$ cannot be a frequent OIP and can be pruned.

Pruning strategy 3 (pruning patterns with size one in *I*-join). If $\mathrm{len}(q_n) \neq 1$, $\mathrm{size}(\mathbf{p}) = 1$, and $e_{nu} \notin I_{\rm pre}[i_{11}]$, then pattern $\mathbf{r}$ cannot be a frequent OIP and can be pruned.

Pruning strategy 4 (pruning patterns with size *m* in *I*-join). If $\mathrm{len}(q_n) \neq 1$, $\mathrm{size}(\mathbf{p}) > 1$, and $e_{nu} \notin S_{\rm pre}[i_{11}]$, then pattern $\mathbf{r}$ cannot be a frequent OIP and can be pruned.

Incremental pattern mining. We propose a global pattern dictionary, denoted as G, to store frequent OIPs. In G, prefix patterns form the keys and inner dictionaries form the values. In an inner dictionary, the keys are tuples consisting of the last item of a pattern and a flag which is either "i" or "s", denoted as (item, flag), and the values represent the supports of the patterns, where if the length of the last itemset of the pattern is one, then the flag is "s"; otherwise, the flag is "i".

Conclusion. We have explored the mining of frequent OIPs in an incremental environment, which employs a forgetting factor to reduce the weights of old batches, and proposed the OIP-Miner algorithm. To verify the performance of OIP-Miner, 16 databases and 13 competitive algorithms were selected. Experimental results show that OIP-Miner is more efficient than other competitive algorithms. More importantly, introducing the forgetting factor improves the confidence rate for OIP-Miner, since users' future behaviors are often more closely related to recent behaviors than previous behaviors. All algorithms can be downloaded from https://github.com/wuc567/Pattern-Mining/tree/master/OIP-Miner.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 62372154, U21A20513, 62076154, 62120106008), National Key Research and Development Program of China (Grant No. 2024YFB3311901), and Open Project of Key Laboratory of Computational Intelligence and Chinese Information Processing, Ministry of Education, Shanxi University (Grant No. CICIP2024008).

Supporting information Appendixes A–D. The supporting information is available online at info.scichina.com and link.springer.com. The supporting materials are published as submitted, without typesetting or editing. The responsibility for scientific accuracy and content remains entirely with the authors.

References

1 Wang J, Huang J. On incremental high utility sequential pattern mining. ACM Trans Intell Syst Technol, 2018, 9: 55

2 Kim S, Kim H, Cho M, et al. Efficient approach for mining high-utility patterns on incremental databases with dynamic profits. Knowl-Based Syst, 2023, 282: 111060

3 Li Y, Ma C, Gao R, et al. OPF-miner: order-preserving pattern mining with forgetting mechanism for time series. IEEE Trans Knowl Data Eng, 2024, 36: 8981–8995

4 Kim H, Yun U, Baek Y, et al. Damped sliding based utility oriented pattern mining over stream data. Knowl-Based Syst, 2021, 213: 106653

5 Geng M, Wu Y, Li Y, et al. RNP-miner: repetitive nonoverlapping sequential pattern mining. IEEE Trans Knowl Data Eng, 2024, 36: 4874–4889