Applied Intelligence (2024) 54:2582–2593
https://doi.org/10.1007/s10489-024-05272-w

Bit-Close: a fast incremental concept calculation method

Yunfeng Ke[1,2] · Jinhai Li[1,2] · Shen Li[1,2]

Accepted: 2 January 2024 / Published online: 19 February 2024
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2024

Abstract
The theory of Formal Concept Analysis (FCA) finds diverse applications in fields like knowledge extraction, cognitive concept learning and data mining. The construction of a concept lattice significantly influences the effectiveness of formal concept analysis; hence, the development of high-performance algorithms for concept construction is crucial. In this paper, we introduce a novel algorithm called "Bit-Close" for formal concept construction. Bit-Close leverages bit representation and operations, fundamental to computer science, to enhance the In-Close algorithm. Furthermore, we explore the parallel method of Bit-Close. Our experimental results, obtained from multiple public and random datasets, demonstrate that Bit-Close outperforms In-Close by approximately 20% and is significantly better than other competing algorithms.

Keywords Formal concept analysis · Construction algorithm · In-Close · Bit-Close · Parallelization

1 Introduction

Formal Concept Analysis (FCA) is a mathematical methodology introduced by Wille [1] in 1982. It facilitates data analysis and rule extraction by constructing concept lattices based on a formal context. Rooted in the principles of partial order sets or lattices, FCA is built on a robust mathematical foundation. Over the past few decades, FCA has experienced substantial growth and has been applied in various domains, including data mining [2], conflict analysis [3], web mining [4, 5], machine learning [6], knowledge discovery [7] and outlier detection [8].

The foundational concept employed in data analysis is the formal concept, which is defined as an ordered pair comprising an object set and an attribute set. Typically, the dataset is structured within the context of formal analysis. By developing a concept lattice, object and attributes can be classified and combined to produce an ordered, structured diagram. Therefore, developing high-performance concept construction algorithms is a crucial task. Concept lattice construction algorithms can be categorized into two primary classes: batch and incremental algorithms. Batch algorithms generate concepts using a top-down or bottom-up approach by incorporating attributes or objects into existing concepts as parent concepts, resulting in the generation of upper-level or lower-level concepts. In contrast, incremental algorithms initiate with an initial concept and subsequently add new objects, updating both newly generated and existing concepts.

To begin, we introduce several representative batch algorithms. Qian et al. [9, 10] proposed a novel method that reduces intent computation and also introduced a decomposition method for constructing the corresponding concept lattice from a formal context. Ma et al. [11] utilized dependence space models to obtain the concept lattice. Ganter et al. [12] introduced the Next-Closure algorithm, which enumerates concept closures using feature vectors of objects or attributes. Lindig et al. [13] presented the UpperNeighbor algorithm, which initiates from the smallest concept, generates analogous parents and ascertains concept generation through a tree structure. This algorithm, however, requires extensive storage space for larger datasets due to the need to maintain complete neighbor relationship informa-

✉ Shen Li
lishen@kust.edu.cn

Yunfeng Ke
keyunfeng@stu.kust.edu.cn

Jinhai Li
jhlixjtu@163.com

1 Faculty of Science, Kunming University of Science and Technology, Kunming 650500, Yunnan, People's Republic of China

2 Data Science Research Center, Kunming University of Science and Technology, Kunming 650500, Yunnan, People's Republic of China

tion. Overall, batch algorithms construct concepts through relationships between concepts and can perform well in static formal contexts. However, batch algorithms require a lot of memory space and are computationally inefficient for large formal context.

Next, we introduce some representative incremental algorithms. Kuznetsov et al. [14] developed the CBO algorithm, which enhances computational efficiency by preventing duplicate concept computation through dictionary order and achieving a pruning effect. Zou et al. [15] proposed an efficient incremental algorithm called FastAddIntent, based on AddIntent [16]. Andrews et al. [17] introduced the In-Close algorithm, which enhances computational efficiency by examining the three cases that arise when the current concept extents intersect with the added attributes. The In-Close family [17, 18] currently represents the best-performing set of algorithms. In general, incremental concept construction algorithms offer advantages, including resource savings and suitability for big data. Hence, incremental algorithms are more popular than batch algorithms. However, the implementation of incremental construction of concept lattices is relatively complex and doesn't always outperform batch algorithms.

To further enhance computational efficiency, numerous parallel algorithms have been proposed, including PCBO [19], FPCBO [20] and CBO implemented using CUDA [21]. Additionally, in response to the diversity of data types, such as fuzzy concept lattices and three-way concept lattices, high-performance algorithms have been introduced. Hu et al. [22] put forword the updating methods of object-induced three-way concept lattices for dynamic formal contexts. Qi et al. [23] introduced a high-performance algorithm for creating three-way concept lattices within a given formal context. Chunduri et al. [24] presented an innovative parallel algorithm for concept generation and the construction of three-way concept lattices, with a focus on knowledge discovery and representation in large datasets. Zhang et al. [25] developed a batch-mode algorithm for direct construction of fuzzy concept lattices by utilizing union and intersection operations on the fuzzy set, scanning the fuzzy formal context only once. Li et al. [26] established a novel method for building an approximate concept lattice within an incomplete context, while Wan et al. [27] introduced a construction method for approximate concept lattices. Inspired by parallel and distributed computing [24, 28–31], we combine bit operations with the In-Close algorithm and introduce the Bit-Close algorithm. This Bit-Close algorithm is based on the framework of the In-Close algorithm and employs bit operations to achieve implicit parallelism. Moreover, it cal also be well parallelized. Subsequently, the experimental results demonstrate the Bit-Close algorithm's marked superiority in computational efficiency, as confirmed through rigorous comparative analysis. Furthermore, we perform a theoretical analysis of Bit-Close and demonstrate that it is more efficient for large data sets.

The paper's structure is as follows: Section 2 introduces formal concept analysis, dictionary ordering and the foundational aspects of bitwise operations. Section 3 outlines the work-flow and pseudo-code of the Bit-Close algorithm. Section 4 includes the preprocessing of the experimental dataset, the introduction of the compared algorithms and the presentation of the experimental results. In Section 5, we delve into a theoretical analysis of the superiority of the Bit-Close algorithm over the other algorithms. Section 6 gives the parallel implementation of the Bit-Close algorithm and conducts a detailed analysis of the parallel effect. Finally, Section 7 concludes the paper and proposes directions for future research.

2 Preminary

The following section will introduce the basic concepts of formal concept analysis, lexicographic order, and bitwise operations.

2.1 Basic concepts of formal concept analysis

Definition 1 [1] The triple (U, G, I) is called a formal context, where $U = \{o_1, o_2, ..., o_n\}$ is a non-empty finite set of objects, $G = \{a_1, a_2, ..., a_m\}$ is a non-empty finite set of attributes, and I is a binary relation on the Cartesian product $U \times G$. $(o, a) \in I$ denotes object o possesses attribute a, and $(o, a) \notin I$ denotes object o does not possess attribute a.

Example 1 Table 1 gives a formal context (U, G, I) with $U = \{o_1, o_2, o_3, o_4, o_5, o_6\}$ and $G = \{a_1, a_2, a_3, a_4, a_5\}$, where number 1 below each attribute indicates that the object has the attribute, and number 0 indicates that the object does not have the attribute.

Usually, the formal context that does not contain empty rows, empty columns, full rows and full columns is called a regular formal context, where "empty" and "full" denote non-ownership and ownership, respectively. Obviously, the formal context of Table 1 is regular.

Table 1 A formal context (U, G, I)

U	a_1	a_2	a_3	a_4	a_5
o_1	0	1	1	0	0
o_2	1	1	0	0	0
o_3	1	0	0	0	0
o_4	0	0	0	0	1
o_5	0	0	0	1	1
o_6	0	0	1	1	1

In order to extract concepts from the formal context (U, G, I), we need to give a formal description of the extent and intent of the concept. First, the following operators are given: $\forall X \subseteq U, Y \subseteq G$,

$$f(X) = \{a \in G : \forall o \in X, (o, a) \in I\},$$
$$g(Y) = \{o \in U : \forall a \in Y, (o, a) \in I\}.$$

That is, $f(X)$ denotes the set of attributes common to all objects in X; $g(Y)$ denotes the set of objects that have all attributes in Y.

2.2 Lexicographic order

Given a set S of words or strings, lexicographic order, denoted as $\prec$, is a binary relationship defined on $S \times S$ such that for any two words or strings ω_1 and ω_1 in S, $\omega_1 \prec \omega_2$ if and only if the following condition holds:

There exists an index i (from left to right) such that the i-th character of ω_1 is before the i-th character of ω_2, and for all j satisfying $1 \leq j < i$, the j-th character of ω_1 is equal to the j-th character of ω_2.

In mathematical notation, the lexicographic order can be presented as(l_{w_1} and l_{w_2} represent the length of w_1 and w_2, respectively):

$$w_1 \prec w_2 \quad \Leftrightarrow \quad \begin{pmatrix} \exists i \text{ such that } 1 \leq i \leq \min(l_{w_1}, l_{w_2}), \\ \text{and } w_1[i] < w_2[i], \\ \text{and } w_1[j] = w_2[j] \text{ for all } j \text{ such that } 1 \leq j < i. \end{pmatrix}$$

Example 2 For the ordered set $(S, \prec)$($\prec$ is the partial ordering symbol) with $S = \{1, 2, 3\}$, the lexicographic order is $(1) \prec (1, 2) \prec (1, 2, 3) \prec (1, 3) \prec (2) \prec (2, 3) \prec (3)$. Here's how we get this order:

Start by comparing the first (leftmost) digit of each number. The numbers beginning with 1 come first. So, we get $\{(1), (1, 2), (1, 3), (1, 2, 3)\}$. Within these, we then look at the next digit (where applicable) to get $(1) \prec (1, 2) \prec (1, 2, 3) \prec (1, 3)$. We then move onto the numbers beginning with 2: $\{(2), (2, 3)\}$. Using the same process, we get $(2) \prec (2, 3)$. Finally, the number beginning with 3 is last: $\{(3)\}$. So, putting it all together, the lexicographical order of the set is: $(1) \prec (1, 2) \prec (1, 2, 3) \prec (1, 3) \prec (2) \prec (2, 3) \prec (3)$.

2.3 Concept dictionary order

As the intent of a concept constitutes a partially ordered set, it is feasible and effective to arrange concepts lexicographically based on their partial order relationships. Given that the intent of a concept represents a partially ordered set, concepts can be organized in a lexicographic manner based

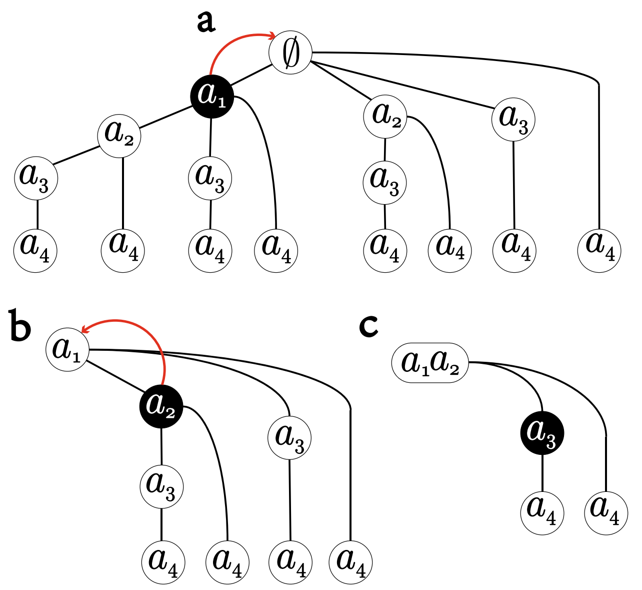


Fig. 1 The attribute set is $\{a_1, a_2, a_3, a_4\}$ and red arrows represent adding corresponding attribute into current intent. $\emptyset$ represents the intent is empty. The left-side arrangement indicates higher precedence in the lexicographic order. **(a)** At this stage, the concept's intent is $\emptyset$, and the arrow signifies the addition of attribute a_1 to the intent, resulting in the updated intent $\{a_1\}$. **(b)** All concepts including attribute a_1 are pruned, forming a new lexicographic tree depicted. The current intent is $\{a_1\}$, and the arrow highlights the inclusion of attribute a_2 in the intent. **(c)** All concepts including attribute a_2 are pruned and the new intent is $\{a_1, a_2\}$

on their partially ordered intent relationships. Assuming the available attributes are $\{a_1, a_2, a_3, a_4\}$, a lexicographic tree can be constructed, as illustrated in Fig. 1(a), wherein the arrangement on the left side indicates a higher precedence in the lexicographic order. At this juncture, the concept's intent is $\emptyset$, which denotes the intent is empty set, with the arrow denoting the addition of attribute a_1 to the intent, resulting in the new intent $\{a_1\}$. Consequently, all concepts preceding a_1 are pruned, yielding a modified lexicographic tree depicted in Fig. 1(b). The concept intent in Fig. 1(b) is $\{a_1\}$, and the arrow signifies the incorporation of attribute a_2 into the intent, generating a new intent and pruning all concepts preceding a_2, as presented in Fig. 1(c).

2.4 Bit operations

Information within a computer is stored as binary numbers in memory, and bitwise operations offer improved computational performance compared to arithmetic operations such as addition, subtraction, multiplication, and division. Numerous bitwise operations exist, however, due to space constraints, this paper will exclusively discuss operations: And, Or, Not, and Left/Right Shifts employed in the experiments. The subsequent examples demonstrate the aforementioned operations, with "B" denoting binary, "D" representing decimal, and "H" signifying hexadecimal.

Left Shift is to shift the binary number to the left by a number of bits, the left (high) part of the shift is rounded off, and the right (low) part is automatically zeroed. For example, for an unsigned number $a = 10110001_B = 177_D = B1_H$, $a \ll 1 = 01100010_B = 98_D = 62_H$.

Right Shift is to shift the binary number to the right by a number of bits, the right (lower) part of the shift is rounded off, and the left (higher) part of the shift is automatically filled with zeroes. For example, for an unsigned number $a = 10110001_B = 177_D = B1_H$, $a \gg 1 = 88_D = 01011000_B = 58_H$.

And is the operation of matching two binary numbers by their corresponding bits, the operation rule is $1\&1 = 1, 0\&1 = 0, 1\&0 = 0, 0\&0 = 0$. For example, for an unsigned number $a = 10011000_B = 152_D = 98_H$, $b = 10000011_B = 131_D = 83_H$, $a\&b = 10000000_B = 128_D = 80_H$.

$$\begin{array}{rr} & 10011000 \\ \& & 10000011 \\ \hline & 10000000 \end{array}$$

Or performs an logical or operation on each corresponding bit of two binary numbers, the operation rule is $1|1$=1, $0|1 = 1$, $1|0 = 1$, $0|0 = 0$. For example, for an unsigned number $a = 11101000_B = 232_D = E8_H$, $b = 00111000_B = 56_D = 38_H$, $a|b = 11111000_B = 248_D = F8_H$.

$$\begin{array}{rr} & 11101000 \\ | & 00111000 \\ \hline & 11111000 \end{array}$$

Not is to take the opposite value of each corresponding bit of a binary number, the operation rule is $\sim 1 = 0$, $\sim 0 = 1$.

$$\begin{array}{rr} \sim & 10110111 \\ \hline & 01001000 \end{array}$$

3 Bit-Close algorithm

The formal context can be viewed as a Boolean matrix consisting of m objects and n attributes. In the process of concept construction, set operations are frequently used. However, set operations on Boolean value sets can perform efficiently through bit operations. From Section 2.4, we can easily find that a bit operation in computers can perform a set comparison of multiple binary bits at the same time. Bit operations have parallel effects. Therefore, inspired by bit operations, we combine bit operations and In-Close algorithm to further improve the efficiency of concept construction.

In this section, we introduce an incremental concept consturtion algorithm called "Bit-Close", which combines the "In-Close" algorithm with bit operations. The main process of the algorithm is shown in Fig. 2, which mainly includes three modules: encoding, calculation concept and decoding.

The binary relationship between objects and attributes is represented as a Boolean matrix $I_{m,n}$, where m and n represent the number of objects and attributes respectively. First, the matrix is encoded to obtain the bitwise stored formal context $I_{m,\lceil n/t \rceil}$, where t represents the bit length. Then, the formal context based on bit storage is used as input to the Bit-Close algorithm to calculate the concept. The Bit-Close algorithm is based on the first concept(full set, empty set) and constructs the concept through the incremental method. Three variables will be maintained: the current attribute j, the index r of the current closed concept and the index r_{new} of the candidate new concept. Finally, all the bit-encoded formal concepts are decoded to obtain the formal concepts. A more detailed description of each module is shown as follows.

3.1 Encoding

To calculate multiple attributes at the same time through bit operations, the original formal context needs to convert into bit storage formation. Figure 3 shows the process of converting the original formal context into bit storage formation.

As shown in Fig. 3(a) and (b), the eight attributes converted into a binary sequence through bit encoding. Figure 3(c) uses hexadecimal to represent this binary sequence. Through bit operations, multiple attributes can be calculated at the same time, thus greatly reducing the number of iterations during the set operation process. Algorithm 1 gives the bit encoding method of formal context.

The storage capacity of binary sequences in computers is limited. Hence, with a large number of attributes, it may be necessary to group attributes and use multiple binary seguences for storage. Tables 2 and 3 provide an example of attribute encoding based on storage length t, typically corresponding to the data type's length. These Tables jointly illustrate the bit encoding process for a formal context with m objects and n attributes. Table 2 represents the original state, while Table 3 depicts the formal context's representation after bit encoding with a storage length of t.

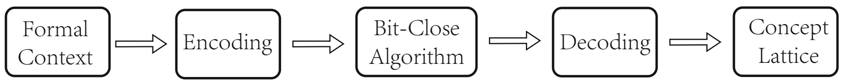
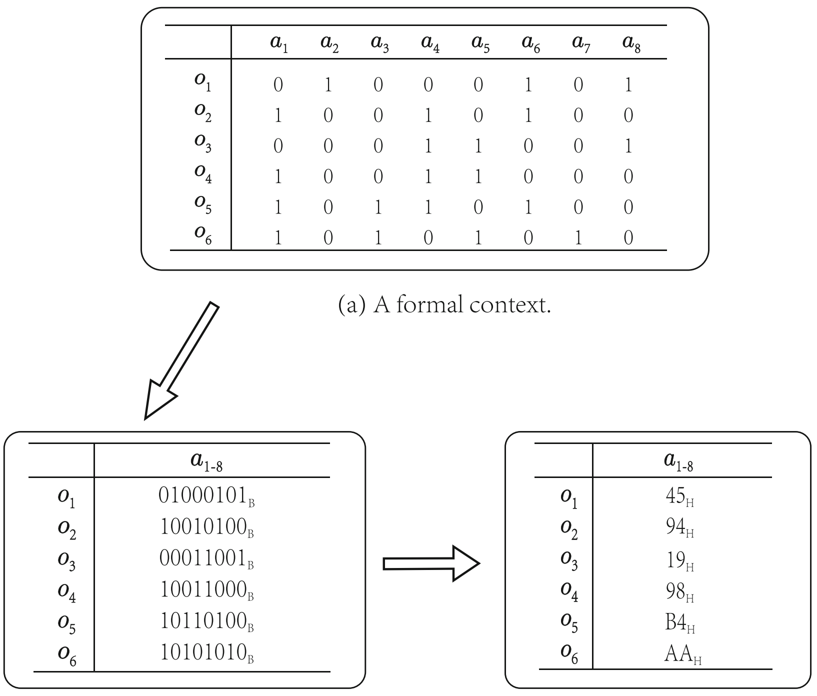


Fig. 2 Incremental construction algorithm flow based on bitwise operations



	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
o_1	0	1	0	0	0	1	0	1
o_2	1	0	0	1	0	1	0	0
o_3	0	0	0	1	1	0	0	1
o_4	1	0	0	1	1	0	0	0
o_5	1	0	1	1	0	1	0	0
o_6	1	0	1	0	1	0	1	0

(a) A formal context.

	$a_{1\text{-}8}$
o_1	01000101_B
o_2	10010100_B
o_3	00011001_B
o_4	10011000_B
o_5	10110100_B
o_6	10101010_B

(b) The binary coded formal context.

	$a_{1\text{-}8}$
o_1	45_H
o_2	94_H
o_3	19_H
o_4	98_H
o_5	$B4_H$
o_6	AA_H

(c) The hex coded formal context.

Fig. 3 Convert a formal context into a binary and hex formal context

Algorithm 1 Bit encoding of the formal context.

```
    Input: Formal Context $I_{m,n}$.
    Output: Formal Context of Bit Coding $I'_{m,\lceil n/t \rceil}$.
1   begin
2       for $i = 0 : m$ do
3           for $j = 0 : n$ do
4               $bit_j = 1 \ll (j \text{ Mod } t)$;
5               $I'_{i,\lfloor j/t \rfloor} = I'_{i,\lfloor j/t \rfloor} \;\&\; bit_j$;
6           end
7       end
8   end
```

3.2 The Bit-Close algorithm

As the formal context undergoes bit encoding, set operations on attribute sets necessitate corresponding algorithms. Algorithm 2 is to add an attribute to an set in the bit-encooed state. Similary, Algorithm 3 is to calculate the difference between two sets in the bit-encoded state.

Algorithm 2 Add attribute j to the set B.

```
    Input: attribute $j$, attribute set $B$.
    Output: $B \cup \{j\}$.
1   begin
2       $Pos_j = \lfloor j/t \rfloor$, $pos_j = j \text{ Mod } t$;
3       $bit_j = 1 \ll (signal[Pos_j] - 1 - pos_j)$ ;
4       $B[Pos_j] = B[Pos_j] \mid bit_j$;
5   end
```

When adding a single attribute to the original set, it must undergo bit encoding before being added through bit operations. In Fig. 4, we present the flow charts of both Algorithms 2 and 3, illustrating their operations through an example. As shown in Fig. 4(a), consider bit-coding the fourth attribute,

Table 2 A general formal context

	a_1	a_2	a_3	a_4	$\cdots$	a_{n-1}	a_n
o_1	0	1	0	0	$\cdots$	1	0
o_2	1	0	0	1	$\cdots$	1	0
o_3	0	0	0	1	$\cdots$	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
o_{m-1}	1	0	1	1	$\cdots$	1	0
o_m	1	0	1	0	$\cdots$	0	1

Table 3 The bit-coded formal context

	a'_1	a'_2	$\cdots$	$a'_{\lceil n/t \rceil}$
o_1	$\text{bit}_{1,1}$	$\text{bit}_{1,2}$	$\cdots$	$\text{bit}_{1,\lceil n/t \rceil}$
o_2	$\text{bit}_{2,1}$	$\text{bit}_{2,2}$	$\cdots$	$\text{bit}_{2,\lceil n/t \rceil}$
o_3	$\text{bit}_{3,1}$	$\text{bit}_{3,2}$	$\cdots$	$\text{bit}_{3,\lceil n/t \rceil}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
o_{m-1}	$\text{bit}_{m-1,1}$	$\text{bit}_{m-1,2}$	$\cdots$	$\text{bit}_{m-1,\lceil n/t \rceil}$
o_m	$\text{bit}_{m,1}$	$\text{bit}_{m,2}$	$\cdots$	$\text{bit}_{m,\lceil n/t \rceil}$

Algorithm 3 Subtraction of attribute sets.

```
Input: The set of attributes $B_1$, $B_2$.
Output: $B_2 - B_1$.
begin
    if $signal[i] == t$ then
        $\widetilde{B}_1 = \sim B_1$;
    else
        $offset = (t - signal[i])$;
        $\widetilde{B}_1 = (\sim B_1 \ll offset) \gg offset$;
    end
    return $B_2$ & $\widetilde{B}_1$;
end
```

resulting in the binary sequence $(0001\ 0000)_B$. By adding this sequence through bit operations to the original set $(1010\ 0000)_B$, we obtain $(1011\ 0000)_B$, effectively adding the attribute to the original set. Similarly, in Fig. 4(b), when calculating the difference between two sets, $B_2 - B_1$, in the bit encoding state, we first invert the B_1 sequence $(0110\ 0011)_B$ to obtain $\widetilde{B}_1 = (1001\ 1100)_B$. Subseouently, we perform the **AND** operation between $\widetilde{B}_1$ and B_2 sequence $(1110\ 0011)_B$, resulting in the binary segeuence of $B_2 - B_1 = (1000\ 0000)_B$.

Consistent with the In-Close algorithm, the Bit-Close algorithm also utilizes a lexicographic approach for implicit searching, avoiding the overhead of computing repeated closures. The Bit-Close algorithm encompasses three rerursive cases:

Case 1: enters recursion by adding a new attribute.
Case 2: quickly obtains corresponding results by checking the intersection of two attribute sets.
Case 3: checks whether the newly calculated concept is newly generated by calling Algorithm 5. If it already exists, it exits the recursion; otherwise, it enters the next recursion branch.

Figure 1 in Section 2 illustrates the process of concept generation, driven by Case 1. As shown in Fig. 1(a) and (b), when adding attributes a_1 or a_2 to the attribute set, the recursion branch is pruned. If the attribute set is empty, it enters Case 2, taking the branch. Otherwise, it enters Case 3, either pruning the branch or completing the recursion. Therefore, the process ensures that concepts are closed only once per concept. Although detecting whether a concept is newly generated reguires iteration, the checking matrix is relatively small. Thus, the concept is generated efficiently.

Algorithm 4 Bit-Close(r, y).

```
begin
    $r_{new}++$;
    for $j = y : n-1$ do
        $A_{r_{new}} = \phi$;
        for $i = 0 : |A[r]| - 1$ do
            if $M[A[r][i]][j]$ then
                $A[r_{new}] \cup A[r][i]$;
            end
        end
        if $|A[r_{new}]| > 0$ then
            if $|A[r_{new}]| == |A[r]|$ then
                $B[r] = B[r] \cup j$;
            end
        else
            if BitIscannonical$(r, j-1)$ then
                $B[r_{new}] = B[r] \cup j$;
                Bit-Close$(r_{new}, j+1)$;
            end
        end
    end
end
```

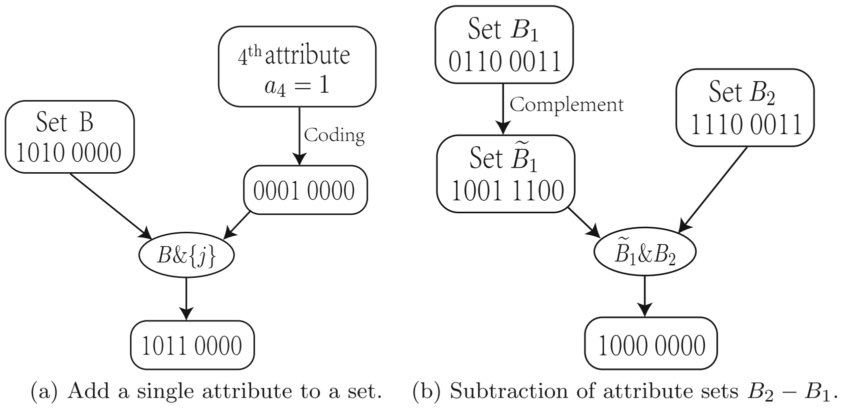
Algorithm 5 BitIscannonical(r, y).

```
begin
    for $i = 0 : \lceil (y+1)/t \rceil$ do
        if $i < \lfloor y/t \rfloor$ then
            $att\_aftery[i] = attrand1[t-1]$;
        end
        if $i == \lfloor y/t \rfloor$ then
            if $(n \text{ Mod } t) == 0$ then
                $att\_aftery[i] = attrand1[y \text{ Mod } t]$;
            else
                $att\_aftery[i] = attrand2[y \text{ Mod } t]$;
            end
        end
        $att\_aftery[i] = att\_aftery[i] - B[r][i]$;
        $intersec[i] = 0 \mid I'[A[r_{new}][0]][i]$;
    end
    for $i = 0 : \lceil (y+1)/t \rceil$ do
        for $j = 0 : |A[r_{new}]|$ do
            $intersec[i] = (intersec[i]$ & $I'[A[r_{new}][j]][i])$;
        end
        $intersec[i] = intersec[i]$ & $att\_aftery[i]$;
        if $0 \mathrel{!}= intersec[i]$ then
            return False;
        end
    end
    return True;
end
```

3.3 Decoding

Since the obtained formal concept is coded, such as $(\{1, 3, 5\}, (0010\ 1000)_B)$, the object set own three objects and the attribute set are bit-coded in a binary sequence. Therefore, we need to decode the binary segeunce of attributes like $(\{1, 3, 5\}, \{3, 5\})$. The pseudo-code of decoding process is proposed in Algorithm 6, which is the inverse of Algorithm 1.

Fig. 4 Bit operations flow chart



Algorithm 6 Decoding.

```
Input: The formal concepts of bit encoding ($\widetilde{A}$, $\widetilde{B}$).
Output: The formal concepts of set representation ($A$, $B$).
1  begin
2      $A = \widetilde{A}$;
3      for $i = 1 : |B|$ do
4          for $j = 1 : \lceil n/t \rceil$ do
5              for $l = 1 : signal[j]$ do
6                  if $B[i][j]$ & $bit_l == bit_l$ then
7                      $B[i] = B[i] \cup \{j \times t + l\}$;
8                  end
9              end
10         end
11     end
12 end
```

4 Experiments

The experimental setup operates on the Windows 10 operating system with the following CPU and memory parameters: an AMD 2700x CPU and 3GB of RAM, respectively. The compared algorithms include In-Close, Add-intent, FCBO, PCBO and IterEss. The experiments involve two categories of datasets: UCI public datasets and randomly generated datasets. Both of these algorithms have been implemented within the Windows C/C++ environment and they utilize the vector container from the C++ Standard TemplateLibrary (STL) to optimize memory usage.

4.1 Compared algorithms

The Bit-Close algorithm's performance is compared with several other algorithms. Here is a more detailed description of these algorithms, with n denoting the number of objects, m representing the number of attributes in the formal context and c signifying the number of generated concepts.

In-Close, a well-known family of algorithms in FCA, is specifically designed for the efficient generation of formal concepts. It overcomes the challenges associated with concept generation by utilizing incremental closure and matrix searching to compute all formal concepts in a formal context swiftly. An outstanding feature of this algorithm is its ability to compute concepts without duplication, ensuring that each concept is counted only once. In-Close is compact, straightforward, requires no matrix pre-processing and is easy to implement. While it demonstrates impressive performance, especially in sparse contexts, its efficiency may be reduced in denser environments. This can be attributed to the inherent characteristics of the algorithm and the challenges posed by dense data structures. The time complexity of the In-Close algorithm is represented as $O(n^2 \times m \times c)$ [32].

AddIntent is an incremental algorithm in FCA designed to build new concepts based on previously established concepts. It reduces redundant concept comparisons by caching already generated concepts. First, the algorithm builds the first concept starting from several previous objects. Then, using this first concept as input, new concepts are generated by continuously merging objects. In contrast to the In-Close algorithm, the AddIntent algorithm explores underlying concepts by recursively traversing the graph, which makes it less efficient. The time complexity of the AddIntent algorithm is expressed as $O(n^2 \times m \times c)$ [32].

FCBO [33], a batch processing algorithm. Through improved normative testing, it tests whether the current concept has been constructed, thereby effectively reducing the repeated generation of formal concepts. Compared to incremental algorithms, batch algorithms can build concepts independently of previous constructions. This feature is particularly beneficial for concept generation in static data sets. FCBO adopts a breadth-first search strategy in operation. First, all adjacent nodes at a given depth are carefully explored before advancing to nodes at subsequent depth levels. However, the disadvantage of the FCBO algorithm is that the amount of calculation is too large. In the worst case, FCBO may degenerate into CBO. The time complexity of the FCBO algorithm is described as $O(m^2 \times n \times c)$ [32].

PCBO is a parallel implementation of the FCBO algorithm, designed to accelerate concept generation through parallel computing. Compared to FCBO, the PCBO algorithm performs faster, especially when using multiple processors. It sequentially explores the top L-level concepts in the primary thread and distributes them among worker threads. The effect of the algorithm is mainly affected by the size of the data set and the distribution ofconcepts. PCBO's time complexity is noted as $O(m^2 \times n \times c)$ [32].

IterEss is an incremental algorithm. It generates formal concepts through a strategy of iterating over important obiects preferentially. The core idea of iterEss is to generate important concepts in the concept lattice structure first and then generate new concepts through iteration based on these basic concepts. It avoids the calculation of unnecessary obiects, resulting in a more efficient and simplified process. The complexity analysis of IterEss is given by $O(m^2 \times n \times c)$ [32].

4.2 Experiments on open datasets

To initiate our experiments, we aimed to select datasets that are both representative and feasible, with approximately ten thousand objects and several hundred attributes. After careful consideration, we chose four datasets from the UCI repository: Mushroom, Nursery, Adult and Letter. These datasets, however, presented several challenges, including multi-valued attributes, real-valued attributes and missing values. To overcome these challenges, we employed a non-zero filling method to handle missing values based on the available data. Additionally, we utilized discretization and one-hot encoding for real-valued data toransform both multi-valued and real-valued attributes into corresponding binary attributes, making them compatible with FCA algorithms.

Adult includes 48842 samples and 15 multi-valued attributes (the number in parentheses corresponds to the number of attributes and the continuous data are discretized): age (16), workclass (9), fnlwgt (16), education (16), education-num (16), marital-status (7), occupation (15), relationship (6), race (5), sex (2), capital-gain (16), capital-loss (16), hours-per-week (16), native-country (42), >50K (2) and there are 4262 missing values in the dataset which are filled, and then the data are one-hot encoded to finally get a the formal context with 200 attributes.

Letter-recognition contains 20,000 samples and 17 multi-valued attributes : capital letter (26), horizontal position of box (16), vertical position of box (16), width of box (16), height of box (16), total pixels (16), mean x of pixels in box (16), mean y of pixels in box (16), x variance (16), y variance (16), x y correlation (16), $x \times x \times y$ (16), $x \times y \times y$ (16), edge count left to right (16), correlation of x-ege with y (16), edge count bottom to top (16) and correlation of y-ege with x (16). The experiments were one-hot encoded on the data to obtain a formal context with 282 attributes.

Table 5 The running time of four datasets

Method	Dataset			
	Mushroom	Nursery	Adult	Letter
Bit-Close	3.193	0.228	19.724	19.004
In-Close [17]	3.256	0.244	22.803	25.593
Add-intent [34]	595.619	440.169	8848.44	3381.96
FCBO [33]	11.859	7.300	596.979	536.993
PCBO [19]	3.397	1.339	244.879	228.768
IterEss [35]	2.234	0.463	26.765	26.683

Nursery includes 12960 samples along with 8 multi-valued attributes : parents (3), has-nurs (5), form (4), children (4), housing (3), finance (2), social (3), health (3) and class (5). The experiments are one-hot encoded on the data to get a formal context with 32 attributes.

Mushroom contains 8124 samples and 23 multi-valued attributes : classes (2), cap-shape (6), cap-surface (4), cap-color (10), bruises (2), odor (9), gill-attachment (2), gill-spacing (2), gill-size (2), gill-color (12), walk-shape (2), walk-root (4), walk-surface-above-ring (4), walk-surface-below-ring (4), walk-color-above-ring (9), walk-color-below-ring (9), veil-type (1), veil-color (4), ring-number (3), ring-type (5), spore-print-color (9) population (6) and habitat (7). There are 2480 missing values in the dataset which are filled and then one-hot encoding is performed on the data to finally get a formal context with 118 attributes.

After completing these preprocessing steps, we obtained formal contexts suitable for analysis by FCA algorithms. The details regarding the size of the formal contexts, the number of concepts and the density of these contexts can be found in Table 4. Furthermore, Table 5 provides the running times of various algorithms on the obtained formal contexts. Notably, the Bit-Close algorithm outperforms others on all four datasets.

Table 4 The processed datasets

Dataset	Mushroom	Nursery	Adult	Letter
The size of formal contexts	8124× 118	12960× 32	48842× 200	20000× 282
The number of concepts	322057	183079	6479149	7677439
Density(%)	19.49	28.13	13.50	6.02

Table 6 The running time with the number of objects ($|G|$) increasing from 200 to 800 with an interval of 200, where $|U| = 10000$ and $p = 0.05$

Method	$\|G\|$			
	200	400	600	800
Bit-Close	0.846	10.459	47.841	131.049
In-Close	1.063	13.048	58.843	169.869
Add-intent	195.647	1297.49	7889.25	21509.6
FCBO	19.620	170.071	739.320	1947.553
PCBO	7.089	94.503	462.426	1494.809
IterEss	5.418	32.724	92.160	206.926

4.3 Experiments on random datasets

In experiments with random data, the number of objects ($|U|$), the number of attributes ($|G|$) and the percentage of non-zero elements in the Boolean matrix to all elements (the density of formal contexts p) were used as control variables to conduct experiments, respectively. The experimental results are shown in Tables 6, 7 and 8.

As demonstrated in Tables 6 and 7, the Bit-Close algorithm exhibits superior performance as the number of objects or attributes increases, respectively. Table 8 reveals that the Bit-Close algorithm outperforms when the density of the formal context is relatively low, but it is not the best choice in high-density contexts. The reason is that, the higher density of non-zero values within a formal concept needs more computations for repetition detection of formal concept through matrix searches. Consequently, this leads to longer running time. Compared with IterEss algorithm, the Bit-Close algorithm is non-parallel, resulting in less efficient of repetition detection. To overcome these challenges, we developed a parallel version of the Bit-Close algorithm in Section 6 and conducted a comprehensive discussion of its parallel principles and effects.

Table 7 The running time with the number of attributes ($|U|$) increasing from 10000 to 40000 with an interval of 10000, where $|G| = 400$ and $p = 0.05$

Method	$\|U\|$			
	10000	20000	30000	40000
Bit-Close	10.196	30.575	52.475	78.351
In-Close	12.945	36.685	67.272	102.027
Add-intent	1456.86	6271	15662.9	31129.5
FCBO	192.886	1637.989	3794.154	8100.822
PCBO	94.503	585.984	864.137	1375.901
IterEss	31.499	98.822	125.924	165.724

Table 8 The running time with the density (p) of formal context increasing from 0.05 to 0.2 with an interval of 0.05, where $|U| = 3000$, $|G| = 150$ and "-" indicates memory overflow

Method	p			
	5%	10%	15%	20%
Bit-Close	0.073	1.283	13.700	127.99
In-Close	0.086	1.632	17.743	177.379
Add-intent	6.825	152.372	2841.84	–
FCBO	0.623	8.619	89.405	653.684
PCBO	0.231	3.360	28.689	208.817
IterEss	0.931	1.299	1.635	2.013

5 Theoretical analysis of Bit-Close algorithm

Given that the Bit-Close algorithm comes from the combination of bit operations and the In-Close algorithm, the theoretical time complexity of Bit-Close and In-Close is the same as $O(|G|^2 \cdot |U| \cdot |C|)$[32], where $|U|$ denotes the number of objects, $|G|$ represents the number of attributes and $|C|$ indicates the number of concepts. However, since Bit-Close implements implicit parallel computing through bit operations, its theoretical running time should be $1/n$ of the In-Close algorithm, where n is the number of attributes in bit storage. Experimental results from both the UCI dataset and random dataset also confirm that, under the same conditions, the Bit-Close algorithm outperforms In-Close. To perform a more detailed comparison of the operational efficiency of each algorithm, a general theoretical model for concept construction algorithm [32] and experiment results are combined to give a quantitative analysis for these algorithms. The running time of the algorithm depends on the number of objects($|U|$), the number of attributes($|G|$) and the density of the concepts(p). The relationship is as follows:

$$time = \alpha_1 \cdot |U|^{\alpha_2} \cdot |G|^{\alpha_3} \cdot p^{\alpha_4} \tag{1}$$

where α_1 is for fitting and does not influence the efficiency of algorithms. The coefficients α_2, α_3 and α_4, respectively, represent the corresponding growth rate of the running time as the number of concepts, the number of attributes, and the concept density increase. For linear regression method based on the experiment results, the running time of the algorithm relationship is transformed into the following formate. The obtained parameters are shown in Table 9.

$$\log(time) = \log(\alpha_1) + \alpha_2\log(|U|) + \alpha_3\log(|G|) + \alpha_4\log(p) \tag{2}$$

From Table 9, we observe that coefficients α_2, α_3, of the Bit-Close algorithm are the smallest among all algorithms. This indicates that, as the number of concepts and attributes

Table 9 The parameter fitting of algorithms

Parameter	Method					
	Bit-Close	In-Close	Add-intent	FCBO	PCBO	IterEss
$\log(\alpha_1)$	-195.782	-241.492	-73226.959	-16491.943	-3872.323	-615.375
α_2	10.503	14.231	8113.605	2901.004	219.758	31.361
α_3	66.797	86.330	6113.963	624.550	585.934	86.111
α_4	87.182	118.898	10013.255	1767.273	387.789	35.203

increases, the increment in the running time of the Bit-Close algorithm is the least pronounced. Additionally, since coefficient α_4 represents the growth rate of the density (p) of the concepts where $p \in (0, 1)$, the impact of density changes on running time is lower than the impact of the increment of objects and attributes. Consequently, Bit-Close algorithm exhibits greater efficiency than other algorithms.

6 The parallel implementation of Bit-Close

Due to the recursive structure of the Bit-Close algorithm, the main challenge is the implementation of parallel recursive computation. Inspired by the parallel implementation of the PCBO [19] and the parallel implement version of In-Close [36], we adopt a similar approach and a detailed description of the Bit-Close parallel algorithm is shown below.

As shown in Fig. 5, a recursive tree formed during constructing formal concepts in Bit-Close algorithm. Therefore, decomposing the recursive tree into several subtrees and assigning them to each thread for calculation is the key to achieving parallel computing. The process is as follows: first determine the maximum recursion depth based on the number of idle threads in the thread pool; next, the recursion tree is decomposed into corresponding recursive subtrees according to the maximum recursion depth and assigned to each thread for simultaneous calculation; when an operation is completed, the corresponding thread uploads the results, enters the thread pool, and requests work; then, repeat the above process until all concepts are constructed.

Figure 6 illustrates the relative running time (i.e. the running time divided by single thread running time) of the Bit-Close parallel algorithm versus the number of threads used on four public datasets. It is obvious that as the number of threads increases, the overall running time gradually decreases. Using dual threads can significantly reduce runtime compared to single threading. However, the reduction in running time is less noticeable when using 8 or 16 threads, probably because the common data set we use is relatively small relative to parallel computing. Too many threads may lead to frequent thread switching, which may reduce operating efficiency. In summary, the Bit-Close parallel algorithm demonstrates efficient parallelization effects.

7 Conclusion

In this study, we introduced Bit-Close, a novel concept construction algorithm that integrates bit operations with

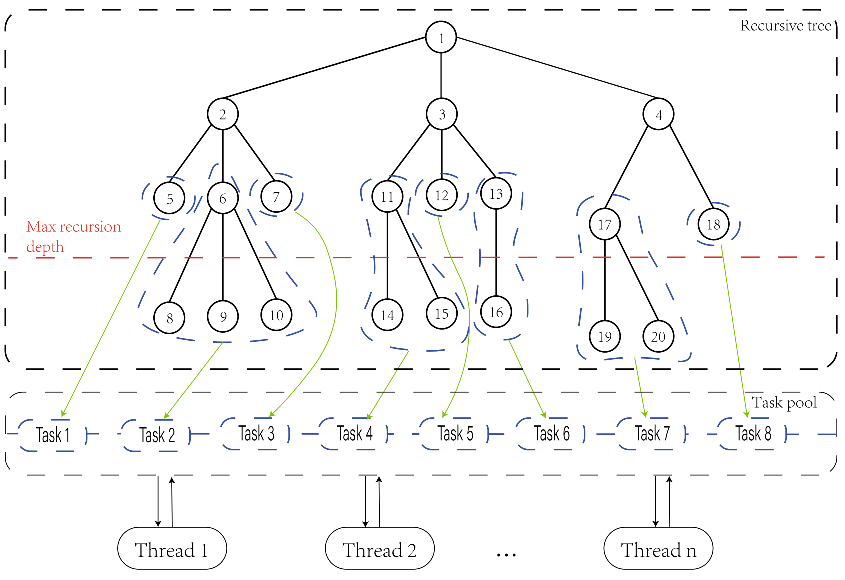


Fig. 5 Parallel calculations on the Bit-Close

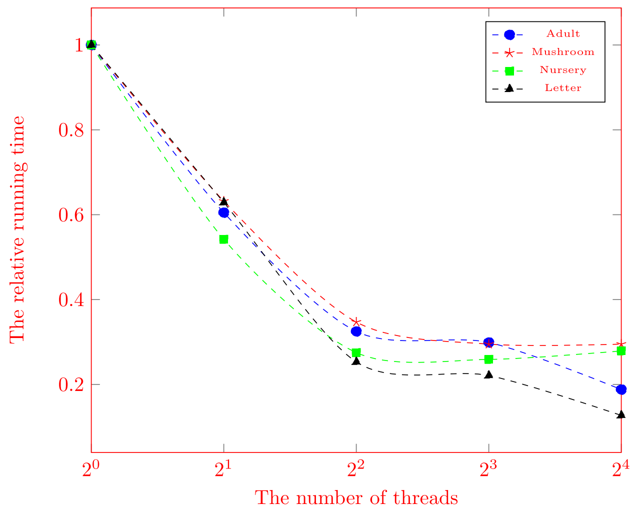


Fig. 6 The relative running time of parallel computing

the In-Close algorithm. Our comprehensive evaluation compared Bit-Close's effectiveness with that of algorithms such as In-Close, Add-intent, FCBO, IterEss and PCBO in formal concept construction tasks. The datasets selected for this assessment, including UCI datasets and randomly generated datasets, varied in size, attributes, and densities. The analysis shows that Bit-Close significantly improves computational efficiency, particularly in larger and denser formal contexts. Additionally, we conducted a theoretical analysis of Bit-Close's advantages over other algorithms. We also investigated Bit-Close's capabilities in parallel computing and validated these through experiments. Future research will aim to adapt Bit-Close for distributed computing within the MapReduce framework, exploiting the framework's efficient memory and resource utilization across multiple devices to enhance parallel computation.

Acknowledgements This work was supported by the National Natural Science Foundation of China (Nos.11971211 and 12171388) and Natural Science Foundation of Yunnan Province (202201AU070136).

Author Contributions Yunfeng Ke: Software, Data curation, Writing - original draft. Jinhai Li: Writing - review & editing, Funding acquisition and discussion. Shen Li: Investigation, Writing - review & editing and discussion.

Data availability statement Data openly available in a public repository. The data that support the findings of this study are openly available in UCI at https://archive.ics.uci.edu/datasets.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Wille R (1982) Restructuring lattice theory: an approach based on hierarchies of concepts. In: Ordered sets. Springer, pp 445–470
2. Thomas J, Cook K (2006) A visual analytics agenda. IEEE Comput Graph Appl 26(1):10–13
3. Wang L, Pei Z, Qin K (2023) A novel conflict analysis model based on the formal concept analysis. Appl Intell 53:10699–10714
4. Jiang F, Fan YS (2010) Web relationship mining based on extended concept lattice. J Softw 21(10):2432–2444
5. Zhang Z, Du J, Wang L (2013) Formal concept analysis approach for data extraction from a limited deep web database. J Intell Inf Syst 41(2):211–234
6. Godin R, Missaoui R, April A (1999) Experimental comparison of navigation in a galois lattice with conventional information retrieval methods. Int J Man-Mach Stud 38(5):747–767
7. Carpineto C, Romano G (1996) A lattice conceptual clustering system and its application to browsing retrieval. Mach Learn 24:95–122
8. Hu Q, Yuan Z, Qin K, Zhang J (2023) A novel outlier detection approach based on formal concept analysis. Knowl-Based Syst 268
9. Qian T, Wei L (2014) A novel concept acquisition approach based on formal contexts. Sci World J 1
10. Qian T, Wei L, Qi J (2017) Decomposition methods of formal contexts to construct concept lattices. Int J Mach Learn Cybernet 8:95–108
11. Ma J, Zhang W, Qian Y (2020) Dependence space models to construct concept lattices. Int J Approx Reason 123:1–16
12. Ganter B (2010) Two basic algorithms in concept analysis. In: Formal concept analysis. Springer, pp 312–340
13. Lindig C, Gbr G (2000) Fast concept analysis. Work Concept Struct - Contrib ICCS 2000:152–161
14. Kuznetsov S (1989) Interpretation on graphs and complexity characteristics of a search for specific patterns. Autom Document Math Linguist 23(1):23–27
15. Zou L, Zhang Z, Long J (2015) A fast incremental algorithm for constructing concept lattices. Expert Syst Appl 42(9):4474–4481
16. Kourie DG, Obiedkov S, Watson BW, Van Der Merwe D (2009) An incremental algorithm to construct a lattice of set intersections. Sci Comput Program 74(3):128–142
17. Andrews S (2009) In-close, a fast algorithm for computing formal concepts. In: ICCS supplementary proceedings. Springer, vol 483
18. Janostik R, Konecny J, Krajča P (2021) Lincbo: fast algorithm for computation of the duquenne-guigues basis. Inf Sci 572:223–240
19. Krajca P, Outrata J, Vychodil V (2012) Parallel recursive algorithm for fca. CLA. Citeseer 2008:71–82
20. Zou L, He T, Dai J (2022) A new parallel algorithm for computing formal concepts based on two parallel stages. Inf Sci 586:514–524
21. Shan B, Qi J, Liu W (2012) A cuda-based algorithm for constructing concept lattices. In: Rough sets and current trends in computing: 8th international conference. Springer, vol 1, pp 297–302
22. Hu Q, Qin K, Yang L (2023) The updating methods of object-induced three-way concept in dynamic formal contexts. Appl Intell 53:1826–1841
23. Qi J, Wei L, Yao Y (2014) Three-way formal concept analysis. In: Rough sets and knowledge technology. Springer International Publishing, pp 732–741
24. Chunduri RK, Cherukuri AK (2023) Distributed three-way formal concept analysis for large formal contexts. J Parallel Distrib Comput 171:141–156
25. Zhang Z (2018) Constructing l-fuzzy concept lattices without fuzzy galois closure operation. Fuzzy Sets Syst 333:71–86
26. Li J, Mei C, Lv Y (2013) Incomplete decision contexts: approximate concept construction, rule acquisition and knowledge reduction. Int J Approx Reason 54(1):149–165

27. Wan Q, Wei L (2015) Approximate concepts acquisition based on formal contexts. Knowl-Based Syst 75:78–86
28. Luo C, Wang S, Li T, Chen H, Lv J, Yi Z (2023) park rough hypercuboid approach for scalable feature selection. IEEE Trans Knowl Data Eng 35(3):3130–3144
29. Luo C, Wang S, Li T, Chen H, Lv J, Yi Z (2023) Rhdofs: a distributed online algorithm towards scalable streaming feature selection. IEEE Trans Parallel Distrib Syst 34(6):1830–1847
30. Luo C, Wang S, Li T, Chen H, Lv J, Yi Z (2022) Large-scale meta-heuristic feature selection based on bpso assisted rough hypercuboid approach. IEEE Trans Neural Netw Learn Syst 1–15
31. Luo C, Cao Q, Li T, Chen H, Wang S (2023) Mapreduce accelerated attribute reduction based on neighborhood entropy with apache spark. Expert Syst Appl 211
32. Kovács L (2018) Efficiency analsyis of concept lattice construction algorithms. Proced Manufac 22:11–18
33. Krajca P, Outrata J, Vychodil V (2010) Advances in algorithms based on cbo. CLA. College Lang Assoc 672:325–337
34. van der Merwe D, Obiedkov S, Kourie D (2004) Addintent: a new incremental algorithm for constructing concept lattices. Concept Lattices. Springer 2961:372–385
35. Outrata J, Vychodil V (2012) Fast algorithm for computing fixpoints of galois connections induced by object-attribute relational data. Inf Sci 185(1):114–127
36. Kodagoda N, Andrews S, Pulasinghe K (2017) A parallel version of the in-close algorithm. In: 2017 6th national conference on technology and management (NCTM). IEEE, pp 1–5

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Yunfeng Ke is pursuing the M.Sc. degree at Kunming University of Science and Technology, Kunming, China. His research interests include formal concept analysis, conceptual cognitive learning, granular computing, and complex network.

Jinhai Li received the M.Sc. degree in science from Guangxi University, Nanning, China, in 2009, and the Ph.D. degree in science from Xi'an Jiaotong University, Xi'an, China, in 2012. He is currently a professor at Kunming University of Science and Technology, Kunming, China. His current research interests include big data, cognitive computing, granular computing, and formal concept analysis.

Shen Li received the Ph.D. degree in science from Institute of Theoretical Physics, Chinese Academy of Sciences, Beijing, China, in 2020. He is currently a lecturer at Kunming University of Science and Technology, Kunming, China. His research interests include rough set, formal concept analysis, cognitive computing and complex network.