Finding all "peaks" of a special "mountain"

Baoqing Jiang[1,2], Wei Wang[2] and Yang Xu[2]

[1]School of Computer and Information Engineering, Henan University, Kaifeng, Henan 475001, China
[2]Center of Intelligent Control and Development, Southwest Jiaotong University, Chengdu, Sichuan 610031, China
Email: jbq@henu.edu.cn

Abstract

In this paper, we discuss a special "mountain": a lower segment of a direct product of finite finite-chains and give an algorithm of finding all "peaks" of the "mountain": all maximal elements of the lower segment. The proposed algorithm is called Boundary algorithm which only tests the elements according to depth-first order in the boundary place of the lower segment and its complementary set. Experiments show that the Boundary algorithm is practicable.

Keywords: Algorithm; direct product; lower segment; maximal element; Data mining

1. Introduction

The problem of mining association rules from large databases has been subject of numerous studies. Some of them focus on developing faster algorithms for the classical method [1] and/or adapting the algorithms to various situations. Another direction is to define rules that modify some conditions of the classical rules to adapt to new applications. Flip Korn focused on real-valued data such as dollar amounts spent by customers on products in a transaction database and proposed *Ratio Rules*. Flip Korn also investigated the application of ratio rules to data cleaning, forecasting, "what-if" scenarios, outlier detection and visualization.

Ratio rules can achieve more compact descriptions and so can better perform extrapolations and predictions, if the data points are linearly correlated. But the data in transaction database are not always like that. So we discuss a weaker form called *Weak Ratio Rules*. In weak ratio rules problem, the transaction database can be thought as a nonnegative real-valued matrix, in which every row corresponds to a transaction and every column corresponds to an item. The weak ratio rules problem is a generalization of Boolean association rules problem. Given minimum support threshold *ms* and minimum confidence threshold *mc*, an important subset *S* of weak ratio rules is just a lower segment of a direct product of finite finite-chains. In order to get all the maximal elements of the lower segment *S*, we design an algorithm Boundary which finds the elements in the boundary place of the lower segment and its complementary set.

In section 2, we will give some basic concepts and results in lattice theory. In section 3, we will propose some new concepts, notations and propositions which certify the correctness of the Boundary algorithm in section 4. For further explaining and testifying the Boundary algorithm, we will give an example and some experimental results in section 5. Finally in section 6, we will give some conclusions and pointers to future work.

2. Preliminaries

We introduce several basic concepts and results which will be used throughout the paper.

Definition 2.1 [2] Let P be a partially ordered set, and S be a subset of P. If every element s in S such that

"$p \le s$ implies $p \in S$", for any $p \in P$,

then S is called a **lower segment** of P. We let empty set $\varnothing$ be a lower segment of P.

By the Definition 2.1, we know that a lower segment consists of all the elements which have a property that if one element has the property then all the elements which are less than or equal to the element have the property. For instance, all large itemsets [1] constitute a lower segment. We can visualize a lower segment as a mountain.

Definition 2.2 [3] Let B be a subset of a partially ordered set P. $b \in B$. If there is no element c in B such that $b < c$, then b is called a **maximal element** of B.

In Definition 2.2, if B is a lower segment, then its maximal elements can be visualized as peaks. Obviously, a finite lower segment can be determined uniquely by its maximal elements.

Let $V_i = \langle v_i[0], v_i[1], \dots, v_i[n_i] \rangle$ be a finite chain, where $v_i[0] < v_i[1] < \dots < v_i[n_i]$, $i = 1, 2, \dots, m$. Let $V \equiv V_1 \times V_2 \times \dots \times V_m$, then an element α in V can be written as $\langle v_1[s_1], v_2[s_2], \dots, v_m[s_m] \rangle$, where $v_i[s_i] \in V_i$, $0 \le s_i \le n_i$ for any $i = 1,2,\dots,m$. For convenience, we call the element α in V as a vector and use $\alpha(i)$ to mean $v_i[s_i]$, the component i of vector α. We know that the direct product V, with the product partial order $\le$ defined as

$\alpha \le \beta$ if and only if $\alpha(i) \le \beta(i)$ for any $i = 1,2,\dots,m$,

is a lattice.

Definition 2.3 Let $\langle v_1[s_1], v_2[s_2], \dots, v_m[s_m]\rangle$ be an element of V, then $\langle s_1,s_2,\dots,s_m\rangle$ is called the **position** of $\langle v_1[s_1],v_2[s_2],\dots,v_m[s_m]\rangle$. And $\langle v_1[s_1],v_2[s_2],\dots,v_m[s_m]\rangle$ is called the **value** of position $\langle s_1,s_2,\dots,s_m\rangle$. The position of v, an element of V, is denoted by $pos(v)$. The value of position p is denoted by $value(p)$. For any element v of V, it is obvious that $v = value(pos(v))$. If for any i=1, 2,…,m, finite chain $V_i=\{0,1,\dots,n_i\}$, then the direct product $V_1\times V_2\times\dots\times V_m$ is called **position lattice**, denoted by Pos($n_1,n_2,\dots,n_m$). For any i = 1,2,…,m, n_i is called i **component upper bound** of Pos($n_1,n_2,\dots, n_m$).

Obviously, the mapping $pos{:}V\to$Pos($n_1, n_2,\dots ,n_m$) and mapping $value$: Pos($n_1,n_2,\dots,n_m$)$\to V$ are reciprocal lattice isomorphisms. Pos($n_1, n_2, \dots , n_m$) is called position lattice of V, denoted by $pos(V)$. For any subset R of V, we use $pos(R)$ to express the set of all positions of elements in R, *i.e.*, $pos(R)\equiv\{pos(v)|v\in R\}$, which is a subset of $pos(V)$. For any subset P of $pos(V)$, we use $value(P)$ to express the set of all values of positions in P, *i.e.*, $value(P)\equiv\{value(p)|\ p\in P\}$, which is a subset of V. If S is a lower segment of V, then $pos(S)$ is a lower segment of $pos(V)$. And obviously we have the equation $value(M(pos(S))) = M(S)$, where $M(S)$ is the set of all maximal elements of S, and $M(pos(S))$ is the set of all maximal elements of $pos(S)$.

By above analysis, the discussion about V can be translated into the discussion about the position lattice $pos(V)$, and the operation on S, a lower segment of V, can be translated into the operation on $pos(S)$ which is a lower segment of $pos(V)$. So, we need only discuss the algorithm of finding all maximal elements of a lower segment of position lattice Pos($n_1, n_2, \dots , n_m$). The position lattice Pos($n_1, n_2, \dots , n_m$) can be thought of as a subset of m-dimensional Euclidean space $\mathbf{R}^m$, and thus a position in Pos($n_1, n_2, \dots , n_m$) is a m-dimensional vector in fact.

3. Theoretical foundation for Boundary algorithm

In the following, suppose S is a lower segment of Pos($n_1, n_2, \dots , n_m$) and $M(S)$ is the set of all maximal elements of S. In order to express the algorithm of finding $M(S)$ conveniently and to demonstrate the correctness of the algorithm, we introduce some concepts and notations.

Definition 3.1 Let $p=\langle s_1, s_2,\dots, s_m\rangle\in$Pos($n_1, n_2, \dots, n_m$), $0\le d\le m$, then the d-dimensional vector $\langle s_1, s_2, \dots , s_d\rangle$ is called ***d*-prefix** of p. When d = 0, the 0-prefix of p is empty vector $\langle\rangle$.

If we imagine an m-dimensional vector p as a string whose length is m, then the d-prefix of p is just the prefix of string p. Like the join of two strings, we can join two vectors and form a new vector. We use $\langle h_1, h_2\rangle$ to express the join of vector h_1 and vector h_2. Obviously, $d(\langle h_1,h_2\rangle) = d(h_1) + d(h_2)$, where $d(h)$ expresses the dimension of vector h which may be empty vector $\langle\rangle$. We let $d(\langle\rangle) = 0$.

Definition 3.2 Let R be a subset of Pos($n_1, n_2, \dots , n_m$), and h be an element of Pos($n_1, n_2, \dots , n_d$), $1\le d\le m$. Then the set of all elements which belong to R and whose d-prefix are h is called ***h*-section** of R, denoted by R_h or $(R)_h$. Let $\langle\rangle$-section of R be itself, *i.e.*, $R_{\langle\rangle} = R$.

The following propositions are theoretical foundation of algorithm Boundary. The proofs are omitted.

Proposition 3.1 Let S be a lower segment of Pos($n_1, n_2, \dots , n_m$), and h be the empty vector $\langle\rangle$ or an element of Pos($n_1, n_2, \dots , n_d$) where $1\le d\le m$. Then

(1). $S_h = (\text{Pos}(n_1, n_2, \dots , n_m))_h \cap S$;

(2). S_h is a lower segment of $(\text{Pos}(n_1, n_2,\dots,n_m))_h$;

(3). $S_h \neq\varnothing$ if and only if $\langle h, \langle 0, \dots , 0\rangle\rangle\in S$ (the number of 0 is m- $d(h)$);

(4). If $s\in S$ and the $d(h)$-prefix of s is greater than or equal to h, then the m-dimensional vector formed by replacing the $d(h)$-prefix of s with h is an element of S_h.

Proposition 3.2 Let S_h be a nonempty h-section of lower segment S, $0\le d(h)\le m$-1. Let U_h denote the set of all m-dimensional vector, in $M(S)$, whose $d(h)$-prefix is greater than or equal to h. If $U_h\neq\varnothing$, let b_0 be the maximum of all components $d(h)$+1 of vectors in U_h. If $U_h = \varnothing$, let b_0 = 0. Then $\langle h, \langle b_0, 0, \dots , 0\rangle\rangle\in S$, where the number of 0 is m- $d(h)$-1.

Proposition 3.3 Let S_h be a nonempty h-section of lower segment S, $0\le d(h)\le m$-1. Let c be the greatest real number of b which satisfy $0 \le b\le n_{d(h)+1}$ and $\langle h,\langle b, 0, \dots , 0\rangle\rangle\in S$. Then for any x: $0\le x\le c$, $S_{\langle h,\langle x\rangle\rangle}$ must be nonempty and $S_h = \cup_{0\le x\le c} S_{\langle h,\langle x\rangle\rangle}$.

By Proposition 3.3, in order to find all the maximal elements, of S, in nonempty h-section S_h, we can find the greatest real numbers c of set

$$\{b \mid 0\le b\le n_{d(h)+1}, \langle h, \langle b, 0, \dots , 0\rangle\rangle\in S\}, \qquad (3.1)$$

then let x decrease from c to 0, and find all the maximal elements, of S, in $S_{\langle h,\langle x\rangle\rangle}$ in turn. The steps (10)–(11) of procedure getmax(h) in the following Algorithm 4.1 can accomplish this task.

How to obtain the maximum c? By Proposition 3.1(3), we know that 0 belongs to the set (3.1). So we can let b = 1 and judge whether b belongs to the set (3.1). If b belongs to the set (3.1), then let b increase 1 until $0\le b\le n_{d(h)+1}$ is false or $\langle h, \langle b, 0,\dots , 0\rangle\rangle\in S$ is false. But this method isn't the best. By Proposition 3.2, we know that b_0 belongs to the set (3.1). Therefore we should let $b=b_0$+1 first. The steps (1)–(8) of procedure getmax(h) in Algorithm 4.1 can accomplish this task.

Actually, when $d(h) = m$-1, we do not need to call the procedure getmax(h) to find all the maximal elements, of S, in $S_{\langle h,\langle x\rangle\rangle}$. In this case, S_h becomes a chain and c is the maximum element of S_h. If $U_h = \varnothing$,

then $\langle h, \langle c\rangle\rangle$ must be a maximal element of S; If $U_h \neq \varnothing$ and $c > b_0$, then $\langle h, \langle c\rangle\rangle$ is a maximal element of S; If $U_h \neq \varnothing$ and $c \leq b_0$, then $\langle h,\langle c\rangle\rangle$ isn't a maximal element of S. The steps (12)–(13) of procedure getmax(h) in Algorithm 4.1 can accomplish this task.

4. Boundary algorithm

The main idea of Boundary is to cut the "mountain" into slices and then, from "higher" slice to "lower" slice, find "all peaks of the mountain" in the slices. Every slice is also a lower segment, so Boundary is recursive.

Algorithm 4.1 Boundary, finding all maximal elements of a lower segment of Pos($n_1, n_2, \dots, n_m$)

Input: natural numbers $n_1, n_2, \dots, n_m$ and
a lower segment S of Pos($n_1, n_2, \dots, n_m$);

Output: M, the set of all maximal elements of S;

Method:

(1) $M := \varnothing$;
(2) if $\langle 0, 0, \dots, 0\rangle \in S$ then
(3) getmax($\langle\rangle$);

procedure getmax(h)

parameter: h, a natural number vector, $d(h)$-prefix of a position in Pos($n_1, n_2, \dots, n_m$);

function: find all the maximal elements, of S, in nonempty h-section S_h and join them into M;

(1) U_h:=the set of all elements, in M, whose $d(h)$-prefix are greater than or equal to h;
(2) if $U_h \neq \varnothing$ then
(3) b_0:=the maximum of all components $d(h)+1$ of vectors in U_h
(4) else $b_0 := 0$;
(5) $b := b_0 + 1$;
(6) while $b \leq n_{d(h)+1}$ and $\langle h, \langle b, 0, \dots, 0\rangle\rangle \in S$ do
(7) $b := b + 1$;
(8) $c := b\text{-}1$;
(9) if $d(h) < m\text{-}1$ then
(10) for $x = c$ downto 0
(11) getmax($\langle h, \langle x\rangle\rangle$)
(12) else if $c > b_0$ or $U_h = \varnothing$ then
(13) $M := M \cup \{\langle h, \langle c\rangle\rangle\}$;

5. Examples and experimental results

The following example can make Algorithm 4.1 understood easily.

Example 5.1 Let S be the following lower segment of Pos($n_1, n_2, \dots, n_m$) (for convenience, $\langle x, y, z\rangle$ is denoted simply by xyz):

$\{xyz \mid xyz \leq 022 \text{ or } 112 \text{ or } 121 \text{ or } 210 \text{ or } 300\}$,

then the process of finding $M(S)$ by Algorithm 4.1 is as follows (only list the test point and the call to getmax(). when we judge whether $p \in S$, we say p is a **test point**):

$M := \varnothing$
$000 \in S$
getmax($\langle\rangle$)
$100 \in S$
$200 \in S$
$300 \in S$
$400 \notin S$
getmax($\langle 3\rangle$)
$310 \notin S$
getmax($\langle 30\rangle$)
$301 \notin S$
$M := M \cup \{300\} = \{300\}$
getmax($\langle 2\rangle$)
$210 \in S$
$220 \notin S$
getmax($\langle 21\rangle$)
$211 \notin S$
$M := M \cup \{210\} = \{300, 210\}$
getmax($\langle 20\rangle$)
$201 \notin S$
getmax($\langle 1\rangle$)
$120 \in S$
$130 \notin S$
getmax($\langle 12\rangle$)
$121 \in S$
$122 \notin S$
$M := M \cup \{121\} = \{300, 210, 121\}$
getmax($\langle 11\rangle$)
$112 \in S$
$113 \notin S$
$M := M \cup \{112\} = \{300, 210, 121, 112\}$
getmax($\langle 10\rangle$)
$103 \notin S$
getmax($\langle 0\rangle$)
$030 \notin S$
getmax($\langle 02\rangle$)
$022 \in S$
$023 \notin S$
$M := M \cup \{022\} = \{300, 210, 121, 112, 022\}$
getmax($\langle 01\rangle$)
$013 \notin S$
getmax($\langle 00\rangle$)
$003 \notin S$

thus, the Algorithm 4.1 is over. There are twenty three test points: 000, 100, … , 003.

Remark 5.1 In Algorithm 4.1, if we replace steps (12)–(13) with the following steps (12)–(16):

(12) else {
(13) $S_0 := S_0 \cup \{\langle h, \langle c\rangle\rangle, \dots, \langle h, \langle 0\rangle\rangle\}$;
(14) if $c > b_0$ or $U_h = \varnothing$ then
(15) $M := M \cup \{\langle h, \langle c\rangle\rangle\}$;
(16) }

position lattice	maximal elements of lower segment	the number of elements in lower segment	the number of test points	runtime (unit: second)
Pos(9,8,7)	⟨0,7,5⟩	48	23	0.031
Pos(9,8,7)	⟨7,0,6⟩	56	31	0.047
Pos(9,8,7)	⟨8,7,0⟩	72	98	0.234
Pos(9,8,7)	⟨0,7,5⟩, ⟨7,0,6⟩, ⟨8,7,0⟩	155	109	0.250
Pos(9,8,9)	⟨6,7,8⟩	504	86	0.219
Pos(9,8,9)	⟨8,7,6⟩	504	104	0.265
Pos(9,8,9)	⟨6,7,8⟩, ⟨8,7,6⟩	616	106	0.266
Pos(9,17,8)	⟨0,0,7⟩, ⟨0,10,0⟩, ⟨8,0,0⟩	26	55	0.078
Pos(9,9,9,9)	⟨1,5,6⟩, ⟨3,2,1⟩	156	68	0.063
Pos(4,3,4,4,4)	⟨0,0,0,2,2⟩, ⟨2,0,2,0,0⟩	17	36	0.078
Pos(10,10,10,10,10)	⟨2,2,2,2,0⟩	81	130	0.344
Pos(43,31,42,43,31)	⟨2,2,2,2,0⟩	81	130	0.359
Pos(10,10,10,10,10,10)	⟨2,2,2,2,1,1⟩	324	294	0.859
Pos(10,10,10,10,10,10,10)	⟨0,0,0,2,2,2,2⟩	81	52	0.125
Pos(10,10,10,10,10,10,10)	⟨2,2,2,2,0,0,0⟩	81	292	0.812
Pos(10,10,10,10,10,10,10)	⟨2,2,2,2,2,2,2⟩	2187	1108	3.531

Table 1. some experimental results

then we can get all elements in S which list in S_0. Of course, the initial value of S_0 is empty set.

Remark 5.2 The function of steps (1)–(8) in procedure getmax(h) of Algorithm 4.1 is to obtain the greatest real number c in Proposition 3.3. Our sequential search method can be replaced with other search method such as binary search.

Some experimental results about Algorithm 4.1 are presented in Table 1 where the lower segments are defined by the method in Example 5.1. By experimental results analysis, we get a general rule: In a general case, the number of test points is less than the number of elements in a lower segment. If most components of maximal elements are zero or most zero components are at the rear, then the number of test points is great than the number of elements in the lower segment. The more the nonzero components are, the longer the runtime is. For the same position lattice, the more test points are, the longer the runtime is. For the same maximal elements, the greater the component upper bounds of position lattice are, the longer the runtime is. In the test point sequence, usually, most elements in the front of it are in the lower segment and most elements in the rear of it are not in the lower segment.

6. Conclusion

We have presented a recursive algorithm Boundary for finding all maximal elements of a lower segment of the direct product of finite finite-chains. Our approach was to slice the "mountain" and then find "all peaks of the mountain" in the slices recursively. In order to express the algorithm conveniently and demonstrate the correctness of the algorithm, we introduced some new concepts, notations and propositions. The experiment results analysis showed the characteristics of our algorithm. In the future we will test the Boundary algorithm on more applications to study its performance further. We are also planning to explore some possible applications in lattice-valued decision-making.

Acknowledgements

The work was supported by the National Natural Science Foundation of P.R. China (No. 60474022) and the Henan Province Natural Science Foundation of P.R. China (No. 2000520025, G2002026). We gratefully acknowledge the valuable and helpful suggestions given by the anonymous reviewers.

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," *Proc. of the 20th Int. Conf. on Very Large Data Bases (VLDB'94)*, pp.487-499. Santiago, Chile, September 1994.

[2] S. Burris and H. P. Sankappanavar, "A Course in Universal Algebra," *Springer-Verlag, New York*, 1981.

[3] Bernard Kolman, Robert C. Busby and Sharon Ross, "Discrete Mathematical Structures," *Prentice-Hall, Inc.*, 1996.