Discover Relaxed Periodicity In Temporal Databases[1]

Tang Changjie Yu Zhonghua Zhang Tianqing
Computer Department, Sichuan University
610064,Chengdu City, China
E-mail: chjtang@scuu.edu.cn

Abstract

The Relaxed-Periodicity Pattern describes loose-`cyclic behavior of objects while allowing uneven stretch or shrink on time axis, limited noises, and inflation /deflation of attribute values. To discover Relaxed-Periodicity from Temporal Databases, we propose the concepts of Attribute Trend, Trend Inertia, Peak-Valley Pattern, Inertia Algorithm with Anti-noise ability, as well as the Peak-Valley Algorithm, and show that the implementation prototype is efficient

.

1 Introduction

In the real world, many objects are associated with time and periodic patterns. To support time in databases, efforts have been put on the designing of Time Related Database such as the 13 Temporal Data Models (TDB) described in [1] and Hbase which is the first TDB prototype implemented in China [2,3]. To discover periodic pattern, some fruitful researches have been made by researchers. Ozden,B. et al. studied the mining of cyclic association rules [4] . Jiawei Han et al. integrated data cube and Apriori data mining techniques for mining segment-wise periodicity in regard to a fixed length period and showed that data cube provides an efficient structure and convenient way for interactive mining of multiple-level periodicity [5]. However, many cycle-like phenomena do not fit strict periodicity because blending with noises and inflation /deflation, or stretching unevenly on the time axis. We refer this kind of phenomena as Relaxed Periodicity. Some previous studies investigated this issue: R. Agrawal et. al. propose fast Similarity search Method in the presence of Noise, Scaling, and Translation[6], Sakoe,H., and Chiba,S proposed dynamic programming algorithm[7] to solve the stretch and shrink in the time axis, but only consider the full matching in voice recognition. Few attentions were paid to mining *Relaxed- periodicity.*

This paper discusses the techniques for mining relaxed-periodicity from *Temporal Databases.* The main contributions include the following:

- Different from previous studies that rely on the every precise attribute value, we propose the *Trend pattern*, including concepts of *Attribute Trend* (such as UP,DOWN, PEAK, and VALLEY), *Trend Inertia* and *Inertia Algorithm* which can ignore small noise and determine the trend of attribute values. The idea is based on the observation that a moving object with inertia can overcome resistance and keep its moving direction.

- With the new concept of *Peak-Valley Chain* , our Peak-*Valley Algorithm* can search relaxed periodicity while disregarding uneven stretch of time axis and inflation/ deflation.

The remainder of the paper is organized as follows. Section 2 gives the concepts related to Relaxed-Periodicity. Section 3 defines the model for Relaxed-periodicity. Section 4 describes our algorithms for mining relaxed periodicity from temporal database. Section 5 discusses the application of relaxed periodicity, such as inflation/deflation. Finally a conclusion is given in Section 6.

2. Preliminaries and Basic Concepts

2.1 The related concepts and operations in TDB

2.1.1 Chronon and Life Span . As defined in [1], *Chronon* is the indivisible time duration supported by temporal system. Time interval is the set of finite ordered chronons $\{t_1,t_2,\ldots t_n\}$ denoted as $[t_1,t_n]$. The length of time

[1] Supported by The National Science Foundation of China No. 69773051

interval is the number of chronons in the interval. The time granularity hierarchy is a user defined series of time interval, such as (Chronon, Second, Minute, Hour, Day, Month, Year). The current time granularity is the smallest time interval to expand temporal information. The *life span* of record (or attribute) is the time interval in which the value of record (or attribute) can be accessed. The function GetLifeSpanBegin(t) return the start time of a life span containing t.

2.1.2 TDrill and Troll. Temporal databases have build-in operations to modify the current time granularity such as *TDrill /Troll* in Hbase, *Expand/Coalesce* in HSQL, *Fold/Unfold* in IXRM and *Nest/Unnest , Pack/Unpack* in other TDB models[1]. They are similar to the Drilling-down/ Rolling-up in data mining, and will be refereed as *TDrill/TRoll*in this paper.

Example 1 Consider a temporal relation with schema <Name, Income, LBegin, LEnd> in Hbase model, where [Lbegin, LEnd] is the life span of record. Give tree records with time granularity of month:

r1= <John, 3000,1975:01:, 1975:01>
r2= <John, 4000,1975:02:, 1975:02>
r3= <John, 5000,1975:03:, 1975:03>.

The operation TRoll-up to SEASON will produce a record r4= <John, 120000, 1975:Spring, 1975: Spring >. The inverse operation TDrill on r4 will Drill-down and get record r1,r2 and r3. ▌

2.1.3 Retrieve Time Related Attribute Value. In temporal database, given time t and attribute A, it is easy to get a record r such that the life span of r contains t, and return the value of r[A]. This retrieval mechanism has various implementation, such as the function Get_Attri_At(A, T) used in this paper .

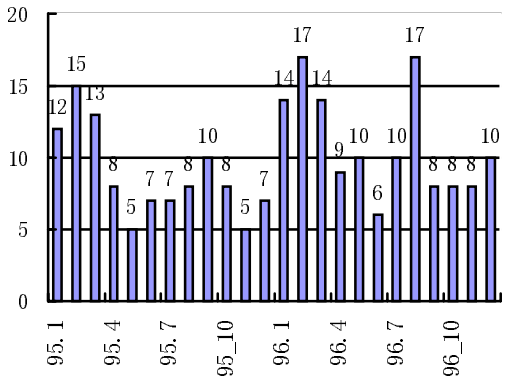

Fig .1 Monthly passengers information (×1000)

Transaction Number	Season	Passengers (×1000)
1	95 Spring	40
2	95 Summer	20
3	95 Autumn	25
4	95 Winter	20
5	96 Spring	45
6	96 Summer	25
7	96 Autumn	35
8	96 Winter	26

Fig.2 Seasonally Passengers information

2.2 The Observations

Following example gives intuitions to *Relaxed Periodicity Pattern.*

Example 2. Consider the passenger information for an airport from January 1995 to December 1996. In the daily routine, the chronon is MINUTE. For the simplicity, we Roll-up to MONTH (as shown in Figure 1) and SEASON (as shown in Figure 2). It is easy to get the following observations:

(1) *The tendency* There are five kinds of tendencies, that is, PEAK (such as Sep.1995, Aug.1996 etc), VALEY (such as May 1995, Sep.--Nov.1996 etc.), UP (such as Jan.1995,Aug,1995 etc), DOWN (such as April 1995) and EdgeInit (such as 1995.1).

(2) *Noises and the Inertia for Trend.* Assume that the system can tolerate noises not greater than 1. Consider the trend at 1996.4 in Fig.1. The point (1996.4,9) is slightly lower than point (1996.5, 10), but the difference is not greater than 1. By assumption, the DOWN trend at April of 1996 will be continued to May of 1996 by *Inertia*. Thus the overall trend for interval [1996.2, 1996.6] is DOWN. The above method to figure out the trend is called *Inertia Algorithm* that will be discussed later.

(3) *Inflation and uneven stretch on time dimension.* Consider the peak points P1="Sep.1995", P3="Aug.1996 " and the valley points V1="May 1995", V3="June.1996 ". The Length from P1 to P3 is 11 months, Length form V1 to V3 is 13 months. Thus there exist uneven shrinks or stretch on time dimension. Comparing the corresponding months of 1995 and 1996, the inflation is clear. ■

3 Model for Relaxed-Periodicity Pattern

To formalize above observations, we now define the Trend , the Time Array and the Trend Array for given attribute**:**

Definition 1. Let [t, t'] be the life span of attribute A, g is the current time granularity, TrendType be the enumerate set {Up, Peak, Down, Valley, EdgeInit }.

(1) The trend function of Attribute A is the map form the Trend of Attribute to Trend Type:

AttriTrend: [t, t'] →TrendType..

Its value AttriTrend(t，A) is called the trend of A at time t.

(2) L= Length([t, t'])/g is called the number of check points for attribute A. The Set

TimeArray={t_k| t_k=t+gk,0<=k<=L}

is called the Time array for attribute A. The set

TrendArray={A_k| A_k = AttriTrend (t_k,A),0<=k<L}

is called the Trend Array of A, its k-th member is denoted as AttriTrend[k].▌

In the real world, objects possess inertia with which it overcomes disturbs (or noises) while trying to keep its original status. For example, a ball rolling down from hill can overcome a small concave and continue rolling down, moreover, the longer duration of rolling, the larger inertia it has. To formalize this observation, we have:

Definition 2. Let [t, t'] and TrendType be as above, and InertiaDom be a subset of real numbers.

(1) The *inertia* function of attribute A is the map from TrendType×$2^{[t,t']}$ *to* InertiaDom:

GetInertia:TrendType×$2^{[t,t']}$ → InertiaDom.

Its value denoted as GetInertia (TheTrend, Durat), where the Durat is the duration of TheTrend.

(2) The inertia evaluation function is the map form TrendType×InertiaDom to NoiseDom, denoted as InertiaToAntiNoise or I_To_N:

I_To_N:: TrendType×InertiaDom→ NoiseDom,

Its value is denoted as I_To_N (TheTrend, InertiaValue).▌

The inertia evaluation function transfer inertia to the the capacity to overcome noises. The separation of two functions (GetInertia. and InertiaToAntiNoise) can increases flexibility in process. The design quality of the functions will influence the mining process.

Example 3. In the Hbase implementation, Users can design their own *GetInertia* function to fit the context of application. The comments give intuitive illustrations.

```
Int Hbase::GetInertia(TheTrend, Duratition )
      // a user-defined inertia function in C++
{ switch (TheTrend)
  { case PEAK : return 0;
    // no inertia at peak.(hence the Peak state is unsteady
    case VALLEY: return 0.2 ;
        //inertia at valley can overcome noise of size 0.2,
        //It is relatively steady.
    case UP: x = Duration; return -a*x^2+bx;
     // a>0,b>0, The inertia of UP trend is a
     //hump-like function. It is increasing by x
     // for x< b/2a, and decreasing by x for b/2a < x.
     //hump-like function is useful for many real objects
    case DOWN: return 0.2 * Ln((Duration);
        //The inertia of down trend
        //is in proportion with the logarithm duration
        //of the trend.
    Default: return EdgeInit;
}}
```

Example 4. This is a simple user-defined inertia evaluation function for Hbase:

```
NoiceDom Hbase::InertiaToAntiNoise(TheTrend,
                                   InertiaValue )
{ swich (TheTrend)
  { PEAK, VALLEY:
          AntiNoiseCapacity = InertiaValue /10;
    DOWN: AntiNoiseCapacity = InertiaValue;
    UP:   AntiNoiseCapacity = 1.1*InertiaValue;
                  // allow 10% inflation
   }   return AntiNoiseCapacity;
  }
```

The trend inertia functions and inertia evaluation function depend on the properties of underlying objects. For example, different stocks may have different inertia functions and inertia evaluation The deep analysis on trend inertia and inertia evaluation functions will be discussed elsewhere.

4. Mining Relaxed-Periodicity From Temporal Database

In the following algorithms, the function SimpleTrend (V1,V2,V3) compares attribute values at continuous time points t1,t2 and t3, and returns the trend at t2. The detail is omitted because the simplicity. The function GetLifeSpanBegin is a typical and simple operation in Temporal Database. Its meaning is as indicated by name and illustrated in Section 2. The function GetInertia and InertiaToAntiNoise are as illustrated in example 3 and 4. The algorithm is written in C++ style for the convenience to illustrate data structure, key programming techniques and loop analysis.

Algorithm 1 (Inertia Trend Algorithm)

Input: Attribute A ,

MaxNoise (the upper bound of noise)

Time array TimeArray[L]，L>=3.

Output: the Attribute trend array TrendArray[L].

```
void HBase::GetTrendWithInertia
    (A ,
    MaxNoise,
    TimeArray)
```

```
{   K=0;  Duration=0;
    PrevTrend = EdgeInit
            // Initialize current trend and duration
 While ( K< L-2){

 // Step1   Get values at three continuous chronons
     for ( J=0; J <=3; J++)
        T [J]= GetLifeSpanBegin (TimeArray[K+J]);
              // get start time of life span
        V[J]=Get_Attr_At(A,T[J]);
              // V[ ] is a buffer array for attribute values
         } //end of for
  if (K=0 && PrevTrend ==EdgeInit)
      // at the first chronon of TimeArray, force the
      // trend as same as the trend at next time point
     {TrendArray[0]=PrevTrend
                    =SimpleTrend( V[0],V[1],V[2]);
           //by trivial algorithm

      continue;   // back to the loop entrance, note
                  //K=0，but PrevTrend   is defined
     } //end of if

 //Step2 :Get and transfer Trend inertia
   PrevInertia= GetInertia(A，PrevTrend , Duration );
                  // Get current inertia, see Example 3
   MaxiNoise +=     // Transfer to anti noise capacity
         InertiaToAntiNoise(PrevTrend, PrevInertia );

//Step 3: Computing current Trend while using inertia
//          as anti-noise capacity.
Switch (PrevTrend )
case   Down:   // It is down at V[0].
   if (( V[0] > V[1]-MaxNoise  ≥  V[2]-2*MaxNoise)
                        //such as Sep.1996 in Fig.1
       || V[0]  ≥  V[1]-MaxNoise > V[2]-2*MaxNoise))
   CurrTrend = Down;
            // keep going down at V[1] by inertia
 else   CurrTrend =Valley; // such as Oct.1996 in Fig.1
       break;

case   Up ://   I it is up at V[0]
    if ( (V[0]  ≤  V[1]+MaxNoise < V[2] + 2*MaxNoise)
             // Noise less than anti-noise capacity
     || (V[0] <V[1]+MaxNoise  ≤  V[2] + 2*MaxNoise))
    CurrTrend   = Up ;   // by inertia, keep going up
    else CurrTrend = Peak ; //V[1] is peak
    break ;

case Valley:
 if   ( V[1]≥V[2] - MaxNoise)) CurrTrend=Valley;
 CurrTrend =Up;   //such as Nov..1996 in Fig.1
 break;

case   Peak: :   .
    if ( V1  ≤  V2 +MaxNoise)) CurrTrend=Peak;
    else CurrTrend = Down;
    break;

otherwise   AskUsersHelp( );
     // complicated noises case, ask user for help
}   // end of   case

TrendArray [K+1]= CurrTrend;
If (PrevTrend== CurrTrend) Duration ++;
      // increate the duration of current status
else Duraion=0; // start a new status
PrevTrend= CurrTrend;
          / / save it , will be used in next loop
 K++;
}   //end of while

//Step 4 : output trend array
TrendArray [L-1]=EdgeInit;
 // the trend at last chronon depends on future's data.

return TrendArray;
}
```

Theorem 1. Let L be the size of TimeArray of attribute A.. Then computing complexity of Algorithm 1 is O(L).

Proof . In the *switch* statement of the Inertia Algorithm (Algorithm 1), each CASE only simply compares attribute values, Although the functions *GetInertia* and *InertiaToAntiNoise* may contain some basic mathematical functions such as logarithm, square root etc. as shown in Example 3 and Example 4, but they are independent from L. Hence the above computation work can be evaluated by a constant C. In the WHILE statement the total loops is L - 2, therefore the total computation complexity is C× (L-2)=O(L). ▌

We now define the chain structure to keep the time series information for all peaks and valleys.

Definition 3 (Peak-Valley Chain). Let A be the attribute concerned, L be the size of the time array of A , and 0<k<L.

(1) The *Peak-Valley Record* of A is the 3-tuple S = (PV, H,T), where H is the value of attribute A at time T, PV= PEAK if the trend of A at T is PEAK , or PV= VALLEY, if the trend of A at T is VALLEY .

(2) The *Peak-Valley Chain* (or *Peak-Valley Pattern)* of Attribute A is the time series chain
$S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_n$
where S_i is called the i-th Peak-Valley Record, and n is the length of chain ▌ .

Following algorithm describes the techniques to pick - up peaks and valleys form Trend Array.

Algorithm 2 (Peak-Valley Algorithm)

Input TrendArray[L], i.e. the output of Algorithm 1.
Output PeakValleyChain[L]. i.e. the Peak-Valley Chain of attribute A.

```
Hbase::Get_Valley_Peak()   //As a member function
                           // of   Hbase
{ int K=0; ListLen=0;
 while ( K<L-1) //copy the data of peak or valley
              //from TrendArray to PeakValleyChain
 { if   (TrendArray [K]== Peak ||
         TrendArray [K]== Valley)
   { PeakValleyChain[ListLen ].PV
           =PeakValleyChain{K}; //is Peak or Valley
     Temp_T= TimeArray[K];
     PeakValleyChain[ListLen ].T= Temp_T;
                                 //as life span start
     PeakValleyChain[ListLen].H=
         Get_Attr_At(A, Temp_T ); //attribute values
     ListLen t++; // count add one
     } //end_of _if
     K++;
 }//end_of_while
 output PeakValleyChain[L];
 }
```

Note that, the size of Peak-Valley chain is much smaller than the size of Trend Array. In Trend Array we only store the trend value, but do not store the attribute value. This saves space. In the Peak-Valley Chain we store the value of attribute, since it is the part of the tendency pattern .

We need a structure to store the aggregation data for all peaks and valleys:

Definition 4. Let the context be as same as Definition 1, MaxPV<L is an integer defined by user as loop control.

(1) Let K>1,S_{j1},S_{j2},…S_{jk} be k continuous peaks (or k continuous valleys). The time difference S_{jk}.T-S_{j1}.T is called k-peak-span (or k-valley-span) started at j1.. The k-peak-span and k-valley-span are also referred as k-span. The set of k-span ordered by the start point *called k-span-array*, denoted as $SpanArray_k$. For coherence we define $SpanArray_k$ ={0} when k=0 and k=1.

(2) For a fixed k>1, denote MaxS = Max($SpanArray_k$), MinS=Min($SpanArray_k$), AveS = Average ($SpanArray_k$) and DiffS= MaxS-MinS. The 4-tuple (MaxS, MinS, AveS, DiffS) is called the *k-span aggregation Record*.

(3) *The Peak-Valley aggregation Array* is defined as PV_AggregateArray[MaxPV], where PV_ *AggregateArray[*k] is the *k-span Aggregation Record*, MaxPV is an user defined constant for loop control.

Theorem 2 (Computing K-Span Array). Let A be the attribute considered. Let L be the number of the check points of A, S_1 ➔ S_2 ➔…➔S_n be the *Peak-Valley Chain* produced by Algorithm 2, and n>2.

(1) If i – j >1 and i - j is an odd number. then S_i.PV = S_j.PV.

(2) The k-span array $SpanArray_k$ can be computed as following:
If k⩽1, $SpanArray_k$[j] = 0. for all j > 0 ,
If k>1, $SpanArray_k$[j] = S_{2k-2+j}T - S_j.T. for j >0.

(3) For n>2,k>1, the number of elements in $SpanArray_k$ is n+2-2k .

Proof (1) The Peak-Valley Chain is constructed based on the attribute trend array. In the step 3 of Algorithm 1, we can see that: DOWN precedes VALLEY, PEAK precedes DOWN, UP precedes PEAK and VALLEY precedes UP. Thus between two VALLEYS there exists at least one PEAK. By the same reason, there exists at least one VALLEY between two PEAKS. Hence the PEAK's and VALLEY's must be one follows another. Thus, S_1.PV = S_3.PV = S_5.PV = ..., and S_2.PV = S_4.PV = S_6.PV =..., as desired. In another word, the output of Algorithm 1 is consistent with our observation to the real world.

(2) The cases for k=0 and k=1 are trivial. Now consider k-span array for k>1. Without loss generality, we may assume S_j.PV is PEAK,. Now we find integer x, such that S_x.PV is PEAK and there are k PEAK's (including S_j and S_x) are between S_1 and S_x. By (1), there are k-1 VALLEY's are between S_j and S_x. Hence x= k +(k-1)+j -1=2k-2+j as desired.

(3) By (2), S_j and S_{2k-2+j} produces on k-span. Note that 1 ⩽j and 2k-2+j⩽n ,hence j⩽n+2-2k. From j=1 to j=n+2-2k, each j is a starting point of a k-span. Thus the total number of k-span in $SpanArray_k$ is n+2-2k. ▌

Example 4. Let S_1 ➔ S_2 ➔…➔S_9 be the *Peak-Valley Chain* produced by Algorithm 2, S_1.PV= S_3.PV= S_5.PV= ….= S_9.PV=PEAK and S_2.PV= S4.PV= S6.PV= S_8 PV=VALEEY. It is easy to verify that

(1) 2-span={ $S_{2x2-2+j}$T - S_j.T | for j >0,j<9}
={ S_3.T-S_1.T, S_4.T-S_2.T, S_5.T-S_3.T
S_6.T-S_4.T, S_7.T-S_5.T,
S_8.T-S_6.T, S_9.T-S_7.T}

(2) 2-span has 9+2-2×2=7 elements.

(3) 3-span={ $S_{2x3-2+j}$T - S_j.T | for j >0,j<9}
={ S_5.T-S_1.T, S_6.T-S_2.T, S_7.T-S_3.T
S_8.T-S_4.T, S_9.T-S_5.T }

(4) 3-span has 9+2-2×3=5 elements. ▌

Theorem 3. The Computing complexity to compute k-span Array is $O(n^2)$,where n is the size of Peak-Valley Chain.

Proof. If n is odd number, it computes one more k-span , (i.e. the case of k= (n+1)/2) than the case of n-1. This does not affect to the complexity, Thus without loss accuracy, we may assume that n is even number and n=2m. By (3) of Theorem 2, the members in the m-span, (m-1)-span,…,2-span are 2,4,…(n+2-4) respectively. The

computing for each span needs one subtraction, the total number of subtractions is 2+4+…+(n-2)=n(n-2)/4. Thus the complexity is $O(n^2)$. ▌

It is easy to see that based on the *Peak-Valley Chain* The Aggregation array PV_AggregateArray[MaxPV] can be constructed by calling simple functions such as Max, Min, Average in a loop. The implementation in Hbase is. G*et_PV_AggregateArray*. Its details were omitted because simplisity. It will be used in following algorithm.

Algorithm 3 (Mining Relaxed-Periodicity)
Input Temporal Database(TDB),
attribute A,
loop control number MaxPV.
Output The Relaxed-periodicity for the values of attribute A in TDB.
Procedure:

- **Step1:** Select appropriate time granularity, Use temporal operation such as projection and TDrill / TRoll, prepare time series data including TimeArray of size L, as the inputs of Algorithm 1.
- **Step2:** By *Inertia Algorithm* (Algorithm1), produce TrendArray[L].for attribute A by inertia principle.
- **Step3 :** By *Peak-Valley Algorithm*, produce *Peak-Valley Chain*
- **Step4**: For k=2 to MaxPV, computing the k-Span Array by Theorem 2. Then construct k-th Peak-Valley Aggregation Record as PV_ AggregateArray[k]. Find integer j such that PV_ AggregateArray[j].DiffS is minimum for $2 \leqslant j \leqslant$ MaxPV.
- **Step5**: Output PV_AggregateArray[k].AveS as Relaxed-periodicity of Attribute A. ▌

Example 5 Mining the Relaxed-periodicity from the data in Figure 2.
(1) Set parameters as following:
The current granularity is MONTH, MaxNoise=1 , loop control number MaxPV=3.
Trend inertia function is:
GetInertia (A , TrendType ， Duration)
=0.01*Duration
Inertia transfer function is
InertiaToAntiNoise (TheTrend, InertiaValue)
= InertiaValue.

(2)Apply our algorithms 1, get the TrendArray as Fig .4
(3) By Algorithm 2, based on the TrendArray and scanning the temporal database ,get the Peak -valley Chain.
The chain is as following:
S1= (Peak, 15, 95. 02) ➔S2= (Valley, 5, 95.05) ➔
S3= (Peak, 10, 95.09) ➔S4= (Valley, 5, 95.11) ➔
S5= (Peak , 17, 96.02) ➔S6= (Valley, 6, 96.04) ➔
S7= (Peak , 17, 96.08) ➔S6= (Valley,8, 96.10)

Time (1995)	Trend	Time (1996)	Trend
95.1	EdgeInit	96.1	Up
95.2	Peak	96.2	Peak
95.3	Down	96.3	Down
95.4	Down	96.4	Valley
95.5	Valley	96.5	Up
95.6	Up	96.6	Up
95.7	Up	96.7	Up
95.8	Up	96.8	Peak
95.9	Peak	96.9	Down
95.10	Down	96.10	Valley
95.11	Valley	96.11	Up
95.12	Up	96.12	EdgeInit

Fig .3 The trend array

(4) For $2 \leqslant k \leqslant$ MaxPV=3, construct k-span array:
the results are:
$SpanArray_2$ = (7,6,5,6,6.4),
$SpanArray_3$ = (12,12,11,11).
By simple aggregation function, get *Peak-Valley Arregation Array*. Its contents is shown in Fig. 4 (with current time granularity = MONTH):

Agg / K	Max-Span	Min-Span	Ave-Span	Diff-Span
2	7	4	5.6	3
3	12	11	11.5	0 .5

Fig. 4 The statistics for k-Span Array

(5) Note that PV_AggregateArray[3] has minimum DiffSpan value. Thus the relaxed periodicity of attribute A is the AveSpan in PV_AggregateArray[3] i.e. 11.5. Werounded 11.5 to 12 (since time granularity is not dividable). Thus the periodicity. of the specified attribute is 12 months. ▌

5 Discussion

(1) Note that our mining process is based on the Peak-Valley Chain (or Peak-Valley pattern) regardless the inflation or deflation. Once we get the Relaxed-periodicity p, we can figure out the average inflation at (k+1)-th period (with respect to k-th period) by the following procedure.

Procedure 2 GetInflation(k) // inflation at (k+1)-th segment

```
{ TotalInflation =0;
   For (i=kp; i<(k+1)*p; i++)
      TotalInflation += (Get_Attr_at(A,k*p+p+i)-
Get_Attr_at(A, k*p+i));
   AverageInflation = TotalInflation / L;
   Return AverageInflation; } ▌
```

(2)We apply our algorithms to the Data Warehouse for Earthquake information of Sichuan Province. The test data keep the transmogrification information for the AnNing River Area measured by stain gauge from 1970 to 1995. The total numeber of records is 10000. The result is consistent with historical facts. The performance study (not included here for lack of space) shows that the implementation of our methods in Hbase is acceptable for temporal databases with practical scale.

6 Summary

We discuss the phenomena with *Relaxed Periodicity*, introduce the concepts of attribute trend, trend inertia and *Peak-Valley Chain*, propose the *Inertia Algorithm* and *Peak-Valley Algorithm.*

Note that, the *Trend Inertia* is the essential property of object. Once we construct a good inertia function and a good inertia evaluation function, we can calculate the trend of temporal attributes just like what happens in the real world. The *Peak-Valley Chain (or Peak-Valley pattern)* disregards the noises and inflation, gives global trend of attributes. The pattern itself is a kind of knowledge, for example, when we talk about a pattern of career "with tree valley and tree peaks, and each peak is higher than previous peak", people may remember a great statesman in the contemporary era of China. The further study on *Peak-Valley Chain, Inertia function and inertia transfer function* will be discussed elsewhere.

References

[1]]Temporal Databases,--Theory, Design and Implementation, A Tansel, J.Clifford, S. Dadia,.Jajodia, A.Segev and R.Snodgrass, The Benjamin Cummings Publishing Company 1993.

[2] Tang Changjie and Xiong Min: " The Temporal Mechanisms in HBase" Journal of Computer Science and Technology, Vol.11, No.4 July 1996, P365—371。

[3] Tang Changjie, Yu Zhonghua, Zhang Tainqing, Xu DaiGangang and Yang Feng: "Storage By Separate Historical Segment in Temporal Database", in Proceedings of The 15-th Conference on Database of China (CDBC 98).Nanjing city, Oct.1998.

[4] Ozden,B., Ramaswamy, S. And Silberschatz, A. 1998. Cyclic association rules, In Proc. Of 1998 international Conference Data Engineering (ICDE'98), p412-421.

[5] Jiawei Han, WanGong anf Yiwen Yin :Mining Segment-wise Periodic Pattern in Time Related Databases, Proc. Of 1998 of International Conf. On Knowledge Discovery and Data Mining(KDD'98) New York City, NY. Aug 1998.

[6] Rakesh Agrawal, King-Ip Ling, Harpreet S. Sawhney Kyuseok Shim, Fast Similarity Search in the presence of Noise, Scaling, and Translation in Time-Series Databases, Procedings of the 21-st VLDB Conference Zurich Switzerland 1995.

[7] Sakoe,H., and Chiba,S.1990, Dynamic Programming Algorithm Optimization for Spoken Word Recognition. In Readings In speech Recognition eds. Waibel,A. And Lee,K., 159- 165.San Mateo,CA: Morgan Kaufman Publishers, Inc.

[8] Usama M.Fayyad,Grerory Piatetsky-Shapiro, Ed. Advanced in Knowledge Discover and Data Mining, AAAI Press The MIT Press 1996.p1-25.