Who's Watching TV? An algorithm for analyzing TV watching sequence

***Abstract*— TV is usually watched by a group of people and TV program service provider collects viewing history from a single TV. It's important and interesting to know how many persons is in front of TV and who's watching the TV now. In this paper, we proposed a novel and efficient algorithm to discover who's watching TV and how many persons in front of TV from TV watching sequence. Experimental results show that our algorithm works efficiently and effectively.**

I. Introduction

Group behavior in website has became more and more common. Lots of users share a common account but behavior in their own preferences. However, such group behavior is a disaster for recommendation system on website especially for E-Commerce website. A family with an single AMAZON account but buy different types of book is a typical example. The recommendation system will mislead by this group behavior. It will recommendation wrong items to the user logining with this account or even reveal others users' private interest to the current user.

Group behavior doesn't appear only in website but also in real life. Usually there is a supermarket shopping card for a family. Family member brings this shopping card to supermarket and purchases his favorite goods. Supermarket date center will record purchasing history for every shopping card for business intelligence propose. However, the recorded history is generated by a group behavior which will lead to wrong recommendation or influence the accurate of business analysis.

Another common group behavior generating scenario is watch TV. Usually TV is shared by family members. The watching sequence comes from one person or multiply persons.For instance, after Dad has watched news report, the son occupies the TV and begin watching Cartoon. The users switch is so smooth without any explicit indication. Even worse, for some famous and interesting shows, the TV is watching by all family members. The preference recorded in history thus becomes extremely noisy. However, with the rapid development of digital television technology, TV watching sequence could be record by large TV program provider and then becomes an extremely valuable data set which could be used for TV show recommendation and census. But group behavior may significantly affect analysis process.

For database perspective, the records generated by group behavior are usually mixing with the normal record in a single table. All records have an unique account ID. So after grouping by the account ID, all records from a single account could be fetched. However, for data mining tasks, mining algorithms or mining experts are interested in real users instead of account users because lots of algorithms and programs are real user based. If the input records are from different users even with the same account ID, these algorithms and programs will never work correctly. For example, the classification algorithms such as C4.5 and Maximum likelihood will be dramatically influenced by different users with single account ID because these classification algorithms will mislead by totally different user preferences.

And by our common sense, we know there isn't a guarantee that there is always an one-to-one correspondence between account user and real user. Thus, splitting history record generated by group behavior will be significant to scientific community.

However, there are no method to deal with history record generated by group behavior. This is not only because the problem is easy to ignore but also the problem is hard to solve. First of all, there are no explicit user switching signal in the sequence. For some observers, They may never discover preference user switching unless we tell them. Secondly, we don't have background knowledge about the users behind this sequence. That is, we don't know how many persons they have and we don't have profiles for each of them. Thirdly, even for human observer, it's hard to judge the preference switching is an appearance of a new user or a current user switching his preference. Last but not least, we even don't have an explicit description for the items they touched.

In this paper, we propose an novel splitting algorithm which will split records generated by an account to corresponding real users. The algorithm will not only tell how many real users hidden in the sequence but also cluster the records to corresponding real users. The preliminary result shows that our splitting algorithm works both efficiently and effectively. In TV watching sequences test, for some sequences our algorithm achieves almost 90% precision and the overall splitting precision is almost 70%. On the other hand, our algorithm works in $O(n^2)$ complexity which would be quite quick in modern person computer.

II. Related Word

Time-series analysis is an active area of research and relates to our watching sequence splitting task.

Discovering sequential patterns was first introduced in [1] and [2]. Especially in [1], the author proposes three algorithms to mine sequential patterns in transaction database. The mined patterns is a maximal sequence with user-specified minimum support.

Jiong Yang et al.[3] proposes a method to calculate asynchronous periodic pattern that may be present only within a subsequence and whose occurrences may be shifted due to disturbance.

The surprising sequential pattern discovery is proposed in[4]. In this paper, the author focus on mining surprising periodic patterns in a sequence of events. The concept of information gain is proposed to measure the overall degree of surprise of the pattern within a data sequence.

Bettini et al.[5] proposed an algorithm to discover temporal patterns in time sequennces. The paper introduces event structures that have temporal constraints with multiple granularities, defines the pattern-discover problem with these structures, and studies effective algorithms to solve it. The basic components of the algorithm includes timed automata with granularities and a number of heuristics.

Xianping Ge et al.[6] proposed a novel and flexible approach based on segmental semi-Markov model to automatically detect specific patterns or shapes in time-series data. The pattern of interest is modeled as a K-state segmental hidden Markov model where each state is responsible for the generation of a component of the overall shape using a state-based regression function.

Jiawei Han et al.[7] developed an efficiet method for mining multiple-level segment-wise periodicity in time-related database by exploring data cube, bit-array, and the apriori mining techniques.

Valery Guranlnik et al.[8] proposed an event detection approach from time series data. The proposed methods uses an iterative algorithm that fits a model to a time segment, and uses a likelihood criterion to determine if the segment should be partition further. Meanwhile, the technique is independent of regression and model selection methods. Experimental results show that the proposed method is more robust than using visual inspection.

An efficient incremental algorithm for identifying distinctive subsequences in multivariate, real-valued time series is described and evaluated in [9]. The application of this algorithm includes financial time series gathered prior to significant declines or advances in the stock market, time series produced by the monitors in an intensive care unit for patients who die, and traces of the behavior of unauthorized users of computer systems.

In [10], the author proposed an suite of methods for mining partial periodicity in time series database. Partial periodicity, which associates periodic behavior with only a subset of all the time points, is less restrictive than full periodicity and thus covers a broad class of applications.

III. Problem Definition

In our discussion, TV watching sequence refers to a channel switching sequence generated from a TV. The switching records in the sequence are arranged according to the switching in time. And between any two switching records, the switching out time of the first records may or may not be the same as the switching out time. If the two times are different, it means the TV isn't watched by any users in this duration. An illustration is depicted in table I.

Channel	Start Time	End Time
Channel 1	22:00:00	22:00:30
Channel 2	22:30:40	23:00:00
Channel 3	23:00:00	23:30:00
...	...	...
Channel 1	23:45:00	23:48:00

TABLE I
TV Watching Sequence Example

Then given a TV watching sequence, the problem is defined as finding how many users have watched the TV and finding the corresponding watching record for each of them. Figure 1 illustrates the problem.

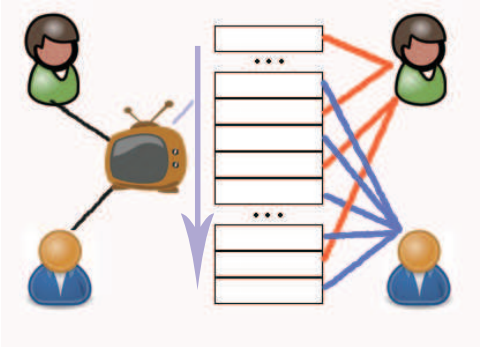


Fig. 1. Problem Illustration

IV. BaseLine Algorithm

The baseline algorithm to finish the splitting task is content-based algorithm. Given the program viewing sequence, content-based algorithm generates similarity matrix whose entry is the similarity between the two programs considered. The algorithm then uses an unsupervised clustering algorithm to cluster the programs. Here, we use hierarchical agglomerative clustering algorithm as the underlying algorithm. After clustering, we could split the sequence into different users. However, we still don't know how many users behind this sequence.

Usually, there are less than 5 persons in a family. So we guess there are $k(1 \leq k \leq 5)$ members in front of TV and force HAC[11] algorithm to generate k clusters. We compare the confidence of the clustering result with regard to different k value. Then, we select the most confident clustering result as splitting result and the corresponding k as the number of members in front of TV. Pseudocode for content-based algorithm is depicted in Algorithm 1

However, there are two main shortcomings of content-based algorithm. First of all, for a single user, he may have multiple interests. For example, a doctor may both like scientific program and news report. However, hierarchical agglomerative clustering algorithm tends to cluster scientific program and news report to different clusters. Thus the algorithm loses

Algorithm 1 Content-based Algorithm

```
1: sim[][] = double[progSize][progSize]
2: userSize = −Infinity
3: maxPurity = −Infinity
4: listOfClusters = NULL
5: for all i ∈ candidateClustersSize do
6:    for all prog1 ∈ progList do
7:       for all prog2 ∈ progList do
8:          sim[indexof(prog1)][indexof(prog2)] =
            contentSimilarity(prog1, prog2)
9:       end for
10:   end for
11:   clusters = HAC(sim[][])
12:   purity = calcPurity(clusters)
13:   if maxPurity < purity then
14:      userSize = i
15:      maxPurity = purity
16:      listOfClusters = clusters
17:   end if
18: end for
```

a lot of accuracy. Secondly, two different users may share a common interest. For example, a doctor and a lawyer may both like news report. But hierarchical agglomerative clustering algorithm usually clusters all news reports into a single cluster. Thus, the algorithm couldn't distinguish the news report belonging to doctor or lawyer.

V. Our Approach

Before we discuss our approach for splitting TV watching sequence, we give some definitions and heuristics first.

Definition 1: $Prob(S|P_1, P_2)$ is the probability that program P_1 and program P_2 viewed by the same viewer.

Definition 2: Bi-gram Program Pair refers to any two adjacent programs in the viewing sequence.

Definition 3: Time Continuity Group refers to a sub-sequence of the viewing sequence such as for any bi-gram program pairs from this sub-sequence the end time of the first program is equal to the start time of the second program.

*Heuristic 1:*If tow programs are watched continuously, then they are likely be watched by a single viewer.

*Heuristic 2:*If two programs are watched continuously and the switching time is the start time of the second program, then they are most probably watched by a single viewer. An illustration is depicted in figure 2

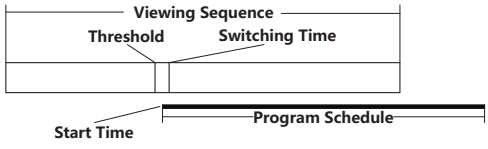


Fig. 2. Illustration of Heuristic 2

A. Mining Prior Knowledge

According to Heuristic 1 and Heuristic 2, we could mine some prior knowledge from given viewing sequences. The first knowledge leant from given viewing sequences is $Prob(S|P_1, P_2)$ if P_1 and P_2 are viewed continuously. The second knowledge leant from given viewing sequences is $Prob(S|P_1, P_2)$ if P_1 and P_2 are viewed continuously and the start time of program P_2 is between the threshold and switching time. The algorithm for mining these two knowledge is depicted in algorithm 2.

Algorithm 2 Mining Prior Knowledge

```
1: bigramCount = 0
2: h1Count = 0
3: h2Count = 0
4: index = 0
5: while index < prog.size() − 1 do
6:    first = prog.get(index)
7:    second = prog.get(index + 1)
8:    inc(bigramCount); inc(index)
9:    if CHECKH1(first, second) == true then
10:      inc(h1Count)
11:      if CHECKH2(first, second) == true then
12:         inc(h2Count)
13:      end if
14:   end if
15: end while
```

Usually, the collected viewing sequences are unlabeled. Thus, before the mining prior knowledge algorithm processing, we need first select some sequences and label them as the training set. After we generate a training set, we could carry out the mining algorithm and get two prior probabilities. The mining task could be repeated for different training set and different time. Averaged result could be treated as prior knowledge.

B. Markov Chain Model

By using prior knowledge, we've already got some useful information to help us finish the splitting task. However, we find the bi-gram program pair which could be used for generating prior knowledge is very few. Yet, we notice that some programs are viewed continuously. For example, program P_1,P_2,P_3 are continuous to each other. From mining prior knowledge algorithm, bi-gram program pairs $<P_1$,$P_2>$ and $<P_2$,$P_3>$ are assigned probability value to help the splitting task. However, by our common sense, we know the program pair $<P_1$,$P_3>$ would also be useful because these two programs are sticked by time continuity. In order to mining probability like $Prob(S|P_1, P_3)$, we came up with a Markov chain model[12][13][14][15].

There is a type of random process which could be characterized as memory-less. We call such random process Markov process. And if the state set is finite we call the Markov process Markov chain.

If the state of random variable X forms a Markov chain, then $P(X_{n+1} = j|X_n = i)$ expresses the probability at time n X is in state i and the next state of i is j. Usually the next state only depends on the current state regardless of time n. Then we could express the probability of state transition as a matrix.

$$\boldsymbol{P} = (p_{ij}) = \begin{bmatrix} p_{00} & p_{01} & p_{02} & \cdots \\ p_{10} & p_{11} & p_{12} & \cdots \\ p_{20} & p_{21} & p_{22} & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix} \tag{1}$$

$p_{ij}^{(n)} = P(X_{m+n} = j|X_m = i)$ expresses the probability the next state of i is j after n step. Here n step means repeating the strategy n times. Then given the initial probability distribution of all objects involved in the strategy, we could find the probability of our interested object.

From mining prior knowledge algorithm, we could know the probability $Prob(S|P_1, P_2)$ if P_1 and P_2 are continuous in time. There are two possible values for $Prob(S|P_1, P_2)$:

$$\begin{cases} Prob(S|P_1, P_2) & = \alpha \; if \; p_1 \; and \; p_2 \; satisfy \; heuristic \; 1 \\ Prob(S|P_1, P_2) & = \beta \; if \; p_1 \; and \; p_2 \; satisfy \; heuristic \; 2 \end{cases} \tag{2}$$

In our method, we use prior knowledge α and β to calculate $Prob(S|P_1, P_2)$ if P_1 and P_2 have time continuity relation between each other but aren't adjacent to each other. That is, we have N programs in the viewing sequence $P_1, P_2, P_3, \cdots, P_n$ and P_i and P_{i+1}($1 \leq i \leq n-1$) are bi-gram program pair. We use Markov chain model to calculate $Prob(S|P_i, P_j)$($i \leq j \leq n$)

We first build the transition matrix of the Markov chain. The matrix is depicted blow. There are two characteristics of the transition matrix. First, according to Markov chain theory, the sum of all probabilities in a row is equal to 1. Secondly, for row i, expect for $Prob(S|P_i, P_i)$ and $Prob(P_i, P_{i+1})$, other probabilities are all zero for we only know the probability of two continuous programs.

$$\boldsymbol{P} = (p_{ij}) = \begin{bmatrix} 1-\alpha & \alpha & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1-\beta & \beta & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1-\beta & \beta & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1-\alpha & \alpha \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{3}$$

After we have built transition matrix, we could compute $Prob(S|P_i, P_j)$ where program P_i and P_j are in the same time continuity group and $i < j \leq n$. We give the formula for calculating $Prob(S|P_i, P_j)$ first and then explain it.

$$Prob(S|P_i, P_j) = V_i \times \boldsymbol{P}^{(j-i)}[j] \tag{4}$$

Here V_i stands for a vector whose i-th column is 1 and other column is 0. $\boldsymbol{P}^{(j-i)}$ stands for the $(j-i)$-step transition matrix and $\boldsymbol{P}^{(j-i)}[j]$ stands for the j-th column of the $(j-i)$-step transition matrix.

V_i stands for the initial probability distribution of n programs. We assume only the i-th program is being viewed now. After $(j-i)$ times program switching, we could calculate the probability that the user is viewing the j-th program. So, according to Markov chain theory, the probability the user at j-th program is $V_i \times \boldsymbol{P}^{(j-i)}[j]$. In other words, the probability concluded from Markov chain theory is same as the probability that program P_i and P_j are viewed by same viewer, namely $Prob(S|P_i, P_j)$.

Noticing the special form of transition matrix, we could simplify the calculation for $Prob(S|P_i, P_j)$. Suppose there are k programs viewed between P_i and P_j($k = j - i - 1$).Then the formulation could be simplified as follows:

$$Prob(S|P_i, P_j) = \prod_{k=i}^{j-1} Prob(S|P_k, P_{k+1}) \tag{5}$$

$Prob(S|P_k, P_{k+1})$ has already given by mining prior knowledge algorithm. Thus we could calculate probability between P_i and P_j if P_i and P_j are in the same time continuity group and $i < j$.

By utilizing Markov chain theory, we expand prior probability to more programs. The simplified formula could give us the probability between two programs in a time continuity group efficiently. Meanwhile, Markov chain theory tells us this probability is reliable. In the following sub-section, we will use the probability which has concluded now to split viewing sequence.

C. Attribute Co-occur Matrix

By using prior knowledge and Markov chain model, we could find $Prob(S|P_1, P_2)$ if program P_1 and P_2 are in the same time continuity group. However, we are interested in $Prob(S|P_1, P_2)$ for each pair of program in the viewing sequence. In order to calculate this probability between any two program, we first introduce the attribute co-occur technique[16][17][18].

The basic idea is that $Prob(S|P_1, P_2)$ could be used as the program attributes' similarity between these two programs. And if we record all attributes' similarity from known $Prob(S|P_1, P_2)$ then this matrix could be used as another prior knowledge.

In our implementation, there're 6 attributes for each program. And for each program, the values for this attribute is obtained from internet by an automatic extraction program. The extraction program will start in the previous weekend and according to downloaded schedule extract values automatically.

For example, we have two attributes A_1 and A_2, and there are N programs containing these two attributes. That is, P_1,P_2,$\cdots$,P_n. Then we could express $Prob(A_1, A_2)$ as follows:

$$Prob(A_1, A_2) = \frac{\sum_{k=1}^{N-1} Prob(S|P_k, P_{k+1})}{N} \tag{6}$$

We need two iteration to generate attribute co-occur matrix. The first iteration extracts all attribute value into a pre-defined attribute list. The second iteration check all program pairs and if the two programs P_1 and P_2 have already been processed in previous knowledge generation module update attribute value pairs from P_1 and P_2 with the probability $Prob(S|P_1, P_2)$. The pseudo-code for building attribute co-occur matrix is in algorithm 3.

Algorithm 3 Building Attribute Co-Occur Matrix

```
1: attrList = NULL
2: attrCo[][] = NULL
3: attrCoCount[][] = NULL
4: for all prog ∈ progList do
5:    for all value ∈ prog.attribute do
6:       if value ∉ attrList then
7:          attrList.add(value)
8:       end if
9:    end for
10: end for
11: while index < progList.size − 1 do
12:    first = progList.get(i)
13:    second = progList.get(i + 1)
14:    for all value_1 ∈ first do
15:       for all value_2 ∈ second do
16:          attrCo[indexof(value_1)][indexof(value_2)]+ =
             Prob(S|first, second)
17:          inc(attrCoCount[indexof(value_1)][indexof(value_2)])
18:       end for
19:    end for
20:    inc(index)
21: end while
22: i = 0; j = 0
23: while i < attrList.size do
24:    while j < attrList.size do
25:       attrCo[i][j]/ = attrCoCount[i][j]
26:       inc(j)
27:    end while
28:    inc(i)
29: end while
```

Attribute co-occur matrix will serve as the final knowledge for the following splitting task. There are two main benefits of using attribute co-occur matrix. The first one is attribute co-occur matrix not only concludes the knowledge we have mined but also expand it to a more expressive and meaningful format. The second one is attribute co-occur matrix are easy to understand both for researchers and for machines.

And there are some alternatives for the range of attribute. The first one is the attribute comes from the sequence considered. And the second one is attribute comes from several sequences. If we use the second one, then we could combine more knowledge together. And if the user behind these sequences are same, then this combination would great benefit the splitting result.

In the next sub-section, we will introduction how to use attribute co-occur matrix to build $P(S|P_1, P_2)$ for each pair of program in the viewing sequence.

D. Probability Model

By using attribute co-occur matrix, we could find $Prob(S|P_1, P_2)$ for any two programs in the sequence. We first give the formula for calculating $Prob(S|P_1, P_2)$:

$$Prob(S|P_1, P_2) = \frac{\sum_{a_1 \in P_1, a_2 \in P_2} Prob(S|a_1, a_2)}{||P_1|| \times ||P_2||} \quad (7)$$

Here, $||P_i||$ stands for the size of attributes for program P_i. And from statistics perspective, equation 7 is the mean of prior knowledge probability for all attribute pairs involved in program P_1 and P_2.

It's easy to understand this definition. The probability the user watches P_1 and P_2 is determined by the probability of attribute pairs from these two programs. And notice that $Prob(S|a_1, a_2)$ is the probability that a_1 and a_2 is viewed by a single viewer. So $Prob(S|P_1, P_2)$ likes a voting result from all attribute pairs.

And if we use alternative 2 to calculate attribute co-occur matrix, then we could get $Prob(S|a_1, a_2)$ from a global perspective. Thus the voting result $Prob(S|P_1, P_2)$ will be more accurate. The pseudo-code for calculating the $Prob(S|P_1, P_2)$ is in algorithm 4.

Algorithm 4 Calculate Probability for Each Program Pair

```
1: prob[][] = double[progSize][progSize]
2: for all prog_1 ∈ progs do
3:    for all prog_2 ∈ progs do
4:       AttrCount = 0
5:       sum = 0
6:       for all a_1 ∈ prog_1 do
7:          for all a_2 ∈ prog_2 do
8:             inc(AttrCount)
9:             sum+ = Prob(a_1, a_2)
10:         end for
11:      end for
12:      prob[indexof(prog_1)][indexof(prog_2)] =
         sum/AttrCount
13:   end for
14: end for
```

By using voting methods to calculate $Prob(S|P_1, P_2)$, we could avoid the problems discussed in baseline algorithm. For example, if a user watches both sports program and news program, then in our baseline algorithm, we could never find similarity between these two programs. However, in our approach, the algorithm could first find some prior knowledge from the viewing sequences which connects sports program and news program and then give us an accurate similarity score for these two programs. And consider another case where both two users like sports program. In our baseline approach, two sports programs will be assigned a high similarity score but they are not watched by a same viewer. And in our approach,

these two programs may be assigned a low probability because we could not find supporting knowledge for these two program. So our approach solve the problem that users may have multi-interests.

E. Clustering

We could use the generated probability $Prob(S|P_1, P_2)$ of each program pair as the input to the clustering algorithm to generate a set of program sub-sequences. The motivation to use clustering algorithm is straight-forward. Consider a simple situation $Prob(S|P_1, P_2)$ has a large value and $Prob(S|P_1, P_3)$ has a small value and $Prob(S|P_2, P_3)$ has a large value too. Then intuitively, we know that program P_1 is very likely viewed by different user from P_2 and P_3. We conclude P_1 is different from P_2 and P_3 instead of different interests because $Prob(S|P_i, P_j)$ is a prior knowledge calculated in the previous section and we know this value has already eliminated multi-interests.

Consider a more complicated example, we have a probability matrix like:

$$\boldsymbol{P} = \begin{bmatrix} 0.8 & 0.8 & 0.1 & 0.1 & 0.8 \\ 0.8 & 0.8 & 0.1 & 0.1 & 0.8 \\ 0.1 & 0.1 & 0.8 & 0.8 & 0.1 \\ 0.1 & 0.1 & 0.8 & 0.8 & 0.1 \\ 0.8 & 0.8 & 0.1 & 0.1 & 0.8 \end{bmatrix} \tag{8}$$

The entry p_{ij} in matrix $\boldsymbol{P}$ is the probability that item i and j belong to the same category. From the matrix, we could see that the probability item 1 and 2 belong to the same category is 0.8 and the probability item 1 and 3 belong to the same category is 0.1. For this matrix, it's easy to see that item 1 2 5 belong to the same category and item 3 4 belong to another category

However, when the matrix becomes very large and the probability is a little fuzzy. It's hard for human to find the corresponding clusters. Thus, we need a clustering algorithm to help us.

Hierarchical agglomerative clustering(HAC)[11] is the fundamental clustering algorithm in our approach. There are two reasons we choose HAC. The first is we only have similarity between two programs and don't have a vector space to represent these programs. The second is that we want to control the number of clusters generated. HAC could fit our requirement.

HAC first treats all programs as a single cluster and then iteratively merges two clusters. In general, the merge operation progress until there is only one cluster remaining. However, we modify the stop condition to fit our requirement. In our approach, when there are N clusters remaining, we stop the merge operation. Figure 3 illustrates the clustering process. There are 5 points in the set originally. And when there are 2 clusters remaining, the clustering progress stops.

Another important thing for HAC is the merge function[19][20][21]. Different merge function could result different clustering result. Here, we use the following formula

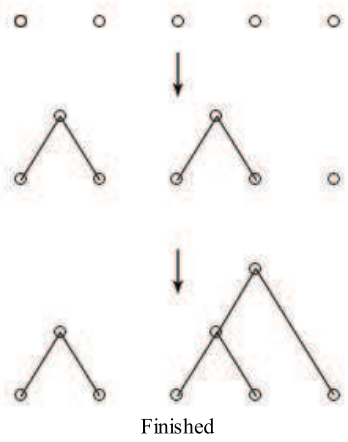


Fig. 3. HAC Illustration

to calculate the similarity between the merged cluster and other clusterings remaining in the candidate cluster list.

$$\begin{aligned} sim_1 &= \frac{(c_i + c_k)}{(c_i + c_j + c_k)} \times sim(i,k) \\ sim_2 &= \frac{(c_j + c_k)}{(c_i + c_j + c_k)} \times sim(j,k) \\ sim_3 &= \frac{c_k}{(c_i + c_j + c_k)} \times sim(i,j) \\ sim((i,j),k) &= sim_1 + sim_2 - sim_3 \end{aligned} \tag{9}$$

Here, c_i refers to the number of points in the cluster i and $sim(i,j)$ refers to the similarity between cluster i and cluster j. Especially, $sim((i,j),k)$ refers to the similarity between the merged cluster(from cluster i and j) and cluster k. We use the this formula to update similarity matrix because we find this updating method could result the most balanced clusters in the general case.

In the last part of this sub-section, we discuss the equivalence between HAC clustering result with our desired splitting result.

First consider c_i, c_j, c_k are equal to 1. Then table II gives all possible cases that the merge process would encounter. In table II, 1 stands for a big value and 0 stands for a small value. For example, if sim(i,k), sim(j,k), sim(i,j) are all big values, then sim((i,j),k) is a big value too. In probability context, if $Prob(S|P_i, P_k)$, $Prob(S|P_j, P_k)$, $Prob(S|P_i, P_j)$ all closes to 1 then program P_i, P_j, P_k are likely viewed by the same viewer($Prob(S|P_i, P_j, P_k)$ closes to 1).

There are some contradictions in table II. For example, the second column tells us program P_i, P_k are likely viewed by the same viewer and program P_j, P_k are also likely viewed by the same viewer but program P_i, P_j are not likely viewed by the same viewer. In this condition, we draw a conclusion that program P_i, P_j, P_k are all very likely viewed by the same viewer. This result is an obvious contradiction to the given factors. But in our approach, this contradiction could never

sim(i,k)	sim(j,k)	sim(i,j)	sim((i,j),k)
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	0

TABLE II

ENUMERATION OF MERGE CASES

happen because we always choose the row whose sim(i,j) = 1(HAC always chooses most similar pairs) and all sim(i,j)=1 row have no contradiction.

When c_i, c_j, c_k are not all 1 but they are equal to each other, we could still use the theory above and thus the cluster result is reasonable.

When c_i, c_j, c_k are not equal, then there are two cases could be discussed. First, cardinality of cluster i or cluster j dominates the weight. Then in such condition, sim((i,j),k) is determined by sim(i,k) if cluster i has dominate weight and by sim(j,k) if cluster j has dominate weight because we always let sim(i,j) be a high value. Second, cardinality of cluster k dominated the weight. Then in such case, sim((i,j),k) is the negative value of sim(i,j) because we don't have enough knowledge for judging the sim((i,j),k).

And in both case, we still guarantee that if program P_1 and P_2 are very likely viewed by a same viewer then these two programs are clustered into a single cluster.

In a conclusion, given the number of viewer and our previous calculated probability matrix, we could find corresponding sub-sequences for each of the viewer by HAC algorithm. The rationality of the clustering result is discussed in the last part of this sub-section. In next sub-section, we'll discuss how to determine the number of viewer in front of TV.

F. Determining the number of viewers

In our approach, we use an enumeration method to determine the number of viewers in front of TV because we assume there are less than 5 viewer in a family watching the TV concurrently.

For each number of viewers, we input this number to the clustering algorithm. Then we will calculate the confidence of the clustering with that input number. That is, for each sequence, we need to run HAC algorithm for 4 times (input number from 2 to 5), and finally we pick the number with the largest confidence as the estimated number of viewers for this sequence. We only need to run clustering algorithm 4 times instead of the whole approach because the probability matrix is common for determining the number of viewer section.

The confidence of the clustering result is calculated in the following way 5.

In this algorithm the function $Sim(P_n, P_m)$ is calculating the similarity between the 2 programs, including not only the content similarity but the time similarity as well. The idea

Algorithm 5 Determining the Number of Viewers

$Count \leftarrow 0$
$Similarities \leftarrow 0$
for all cluster C_i in the split result **do**
  for all cluster C_j other than C_i **do**
    for all program $P_n \in C_i$ and $P_m \in C_j$ **do**
      $Count \leftarrow Count + 1$
      $Similarities \leftarrow Similarities + Sim(P_n, P_m)$
    end for
  end for
end for
$Confidence \leftarrow Similarities/Count$

of this method is base on the fact that the more dissimilar the programs in different clusters are, the more likely we are splitting the programs watched by different users correctly. We have once considered using inner similarity as confidence, where we regard higher similarity within each cluster as higher confidence. Later we find it will always select the larger number of viewers because more clusters lead to higher similarity, while the outer similarity approach we are using now don't have such problems. We will illustrate this further in the evaluation part.

VI. EXPERIMENTAL RESULT

The prilimanary result

A. Best Match

As we have got the clustering result, a best match method given below is deployed to evaluate the precision of the split.

Algorithm 6 Best match evaluation algorithm

Require: n clusters of programs as the result R, m sets of programs watched by the m people as the ground truth T
Ensure: The best match precision of the split
if $n \geq m$ **then**
  $T \Leftarrow T$ add $(n - m)$ empty sets
else
  $R \Leftarrow R$ add $(m - n)$ empty sets
end if
$precision \leftarrow 0$
for all bijection f from R to T **do**
  $cnt \leftarrow 0$
  for all cluster r in R **do**
    $cnt \leftarrow cnt + numberOfIntersections(r, f(r))$
  end for
  if $match > precision$ **then**
    $precision \leftarrow match$
  end if
end for
return $precision$

Under such evaluation, the precision is 100% when everything gets right. If we do not predict the number of users

correctly (more or less users are detected), we will be charged with a heavy penalty that some empty sets are involved such that no result will match on them, which has a large impact on the result.

For instance, given a sequence watched by 3 different people who respectively watched {A, B}, {C, D}, {E, F, G} programs, if our algorithm predicts it is watched by 2 people with the clusters {A, B, C} and {D, E, F, G}, the precision will be 71.4% where A, B, E, F, G are matched in the best match.

B. Experiment Setup

We tested our approach and baseline method in 16 viewing sequences. The 16 viewing sequences consist of 4 groups with 4 sequences in each group. The group is divided by the number of viewer behind the sequence. The distribution of this 16 viewing sequences is depicted in figure 4.

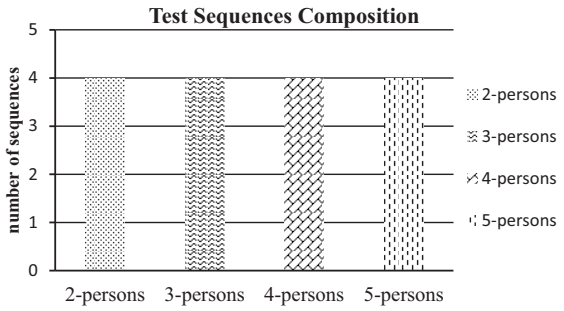


Fig. 4. Test Sequences Illustration

We test our approach for each of the test sequence.

C. Experiment Result

1) Precision of Splitting the Watching Sequence: In figure 5, we list the comparison results of our approach and baseline approach. The comparison is from different perspective. Figure 5(a) shows that for the majority of the test sequence our approach over-performs the base-line approach. Figure 5(b) shows that for different group size our approach over-performs the base-line approach.

2) Precision of Determining the Number of Viewers: In the previous section we have talked about the algorithm 5 that we use to determine the number of viewers behind the watching sequence. We conduct the experiment twice, using the baseline method and the attribute co-occur method respectively. And we compare the result of n to both the real number of viewers and the number with the best precision.

The result shows in 23 of the 32 sequences we are selecting the n with the best precision, the ratio is 72%, and in 18 of the 32 sequences we are selecting the n which is correct according to the real value, the ratio is 56%. The distribution of the n selected is give below in graph 6.

This result infers that even if the number of viewers behind the sequences were known in our problem, we were not going to reach the best precision, because in some situation the misunderstanding of the number of viewers can lead to a better result.

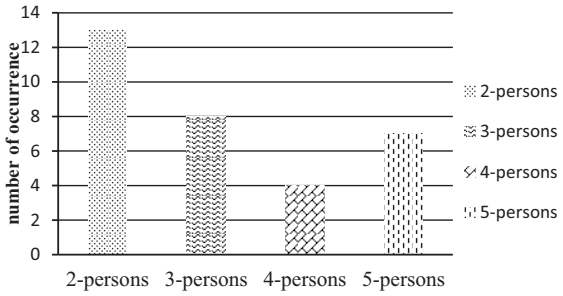


Fig. 6. Experimental result of Determining the number of viewers algorithm

This partially results from the case where some watchers are watching programs of multiple styles that may exactly match the style of some other viewers with relatively pure styles.

VII. Conclusion

In a conclusion, we proposed a novel TV viewing sequence splitting algorithm based on Markov Chain Model. The experimental results show that our approach works both efficiently and effectively. Meanwhile, the proposed approach could be used to analyze more histories by modifying the model a little. The future research direction could be making the algorithm more accurate and eliminating the prior knowledge mining process.

References

[1] R. Agrawal and R. Srikant, "Mining sequential patterns," in *ICDE*, 1995, pp. 3–14.

[2] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *EDBT*, 1996, pp. 3–17.

[3] J. Yang, W. Wang, and P. Yu, "Mining asynchronous periodic patterns in time series data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 15, no. 3, pp. 613 – 628, may-june 2003.

[4] J. Yang, W. Wang, and P. S. Yu, "Infominer: mining surprising periodic patterns," in *KDD*, 2001, pp. 395–400.

[5] C. Bettini, X. Wang, S. Jajodia, and J.-L. Lin, "Discovering frequent event patterns with multiple granularities in time sequences," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 10, no. 2, pp. 222 –237, mar/apr 1998.

[6] X. Ge and P. Smyth, "Deformable markov model templates for time-series pattern matching," in *KDD*, 2000, pp. 81–90.

[7] J. Han, W. Gong, and Y. Yin, "Mining segment-wise periodic patterns in time-related databases," in *KDD*, 1998, pp. 214–218.

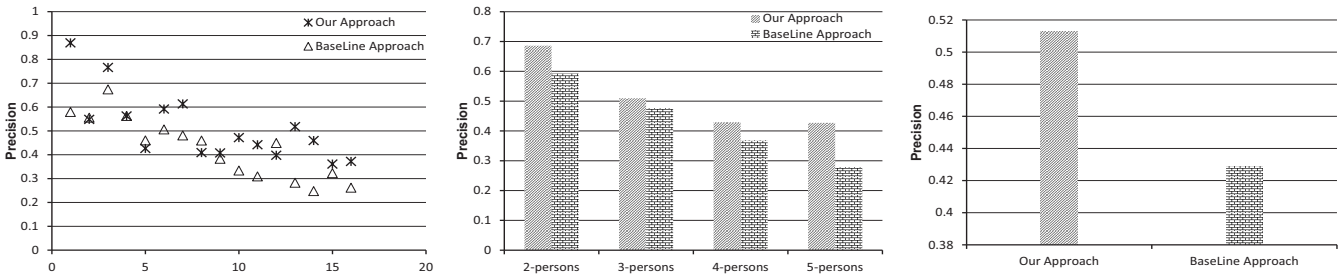
[8] V. Guralnik and J. Srivastava, "Event detection from time series data," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '99. New York, NY, USA: ACM, 1999, pp. 33–42.

[9] T. Oates, "Identifying distinctive subsequences in multivariate time series by clustering," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '99. New York, NY, USA: ACM, 1999, pp. 322–326.

[10] J. Han, G. Dong, and Y. Yin, "Efficient mining of partial periodic patterns in time series database," in *Data Engineering, 1999. Proceedings., 15th International Conference on*, mar 1999, pp. 106 –115.

[11] R. Gil-García, J. M. Badía-Contelles, and A. Pons-Porrata, "A general framework for agglomerative hierarchical clustering algorithms," in *ICPR (2)*, 2006, pp. 569–572.

[12] Y. Ephraim and N. Merhav, "Hidden markov processes," *Information Theory, IEEE Transactions on*, vol. 48, no. 6, pp. 1518 –1569, jun 2002.



(a) Precision Comparision by each point (b) Precision Comparision by group size (c) Overall Presicion Comparison

Fig. 5. Experimental Result

[13] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257 –286, feb 1989.

[14] L. Rabiner and B. Juang, "An introduction to hidden markov models," *ASSP Magazine, IEEE*, vol. 3, no. 1, pp. 4 –16, jan 1986.

[15] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-checking algorithms for continuous-time markov chains," *IEEE Trans. Softw. Eng.*, vol. 29, no. 6, pp. 524–541, June 2003.

[16] B. He and K. C.-C. Chang, "Automatic complex schema matching across web query interfaces: A correlation mining approach," *ACM Trans. Database Syst.*, vol. 31, no. 1, pp. 346–395, Mar. 2006.

[17] X. Li, X. Wu, X. Hu, F. Xie, and Z. Jiang, "Keyword extraction based on lexical chains and word co-occurrence for chinese news web pages," in *Data Mining Workshops, 2008. ICDMW '08. IEEE International Conference on*, dec. 2008, pp. 744 –751.

[18] M. Celik, S. Shekhar, J. Rogers, J. Shine, and J. Kang, "Mining at most top-kmixed-drove spatio-temporal co-occurrence patterns: A summary of results," in *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, april 2007, pp. 565 –574.

[19] R. B. Zadeh and S. Ben-David, "A uniqueness theorem for clustering," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, ser. UAI '09. Arlington, Virginia, United States: AUAI Press, 2009, pp. 639–646.

[20] G. Carlsson and F. Mémoli, "Characterization, stability and convergence of hierarchical clustering methods," *J. Mach. Learn. Res.*, vol. 99, pp. 1425–1470, August 2010.

[21] K. Li, L. Wang, and L. Hao, "Comparison of cluster ensembles methods based on hierarchical clustering," in *Computational Intelligence and Natural Computing, 2009. CINC '09. International Conference on*, vol. 1, june 2009, pp. 499 –502.