Finding TAPs using sequence mining algorithm

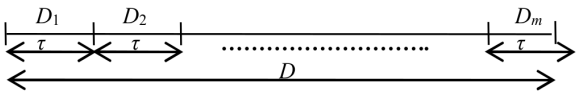
Anmol Arora

November 16, 2016

1 Problem Statement

The entities involved in many chronologically ordered transactional data logs fall into two disjoint classes such as sellers and buyers, customers and items, etc. In many of these data logs the transactions among same pair of entities recur, e.g., a customer buying an item regularly. In these entity-to-entity temporal (EET) logs, the recurring relationship between the entities may expose interesting temporal patterns in the data, which are henceforth reffered to as TAP(Temporal Association Pattern).

To start with, let us suppose that the data available with us is in the form of transactions between entities of two classes. These transactions are chronologically ordered and are associated with labels that identify the characteristics of these transactions. A hyperparameter τ (quantization parameter) is defined which specifies the temporal granularity at which the temporal association between the entitities is tracked. The EET log is divided into a sequence of quanta of duration τ. Each quantum contains log records from a non-overlapping time period of τ time units.



(a) EET log divided at time granularity τ

Definition 1.1. Temporal Association Pattern (TAP) A TAP P of length n, starting in quantum k, is an ordered sequence of states, $P = \prod_{i=1}^{n} S_i : k | S_i \in \{S_1..S_S\}, \beta \leq n \leq m; min(|P.u|, |P.v|) \geq \alpha$

Entities $P.u \in U$ and $P.v \in V$ follow the sequence of states represented by pattern P. $P.u$ and $P.v$ are the support sets of P. α is the size of support of P and β is the minimum length of patterns. The hyperparameters α, β are decided by the user beforehand.
Thus, our objective is to identify all TAPs of length $\geq \beta$ with support of at least α, i.e., $min(|u|, |v|) \geq \alpha$
The framework for solving this problem efficiently has been described in the paper(in progres) titled "Discovering Inter-Class Temporal Association Patterns"[Manoj Agarwal, Krithi Ramamritham]. Their method finds all TAPs, of given minimum length and support, in $O(ElogE_{max})$, where E is the total number of edges and E_{max} is maximum number of edges over all bipartite graphs.
However this method has not been benchmarked against other more commonly known generic data

mining algorithms. Infact it is not even established if other such algorithms can be used to solve this problem of finding TAPs.
Specifically the problem this paper tries to solve is:

Problem Statement. *To verify the existence and development of a framework which transforms an instance of finding TAPs to an equivalent instance of some other data mining problem(Sequential Pattern Mining has been chosen as the candidate benchmarking algorithm for this paper) and experimentally compare their performance.*

2 Research Contributions

- Designed a framework which translates the problem of finding TAPs into a generic sequential pattern mining problem. This framework has been discussed in Section 3.1
- Implemented the above mentioned framework in C++ and solved the problem of finding TAPs through sequence mining algorithm(specifically PrefixSpan), discussed in Section 4.1
- Empirical results comparing the performance of the two algorithms, discussed in Section 4.2

3 Translation Framework

Given the data log, projection function π (σ , T, D_{raw}), quantization parameter τ and the state mapping function ψ, we can show that one can extract all continuous TAP sequences using sequence mining algorithm(like *PrefixSpan*).

3.1 Algorithm and Proof

- Apply projection, quantization, aggregation and state mapping on the EET log.
- The assumption is that the transactions occur between two entities U and V. Consider the ordered transactions over quanta between any pair u, v : u ∈ U, v ∈ V as sequences.
  For example if u, v have edges labelled as:

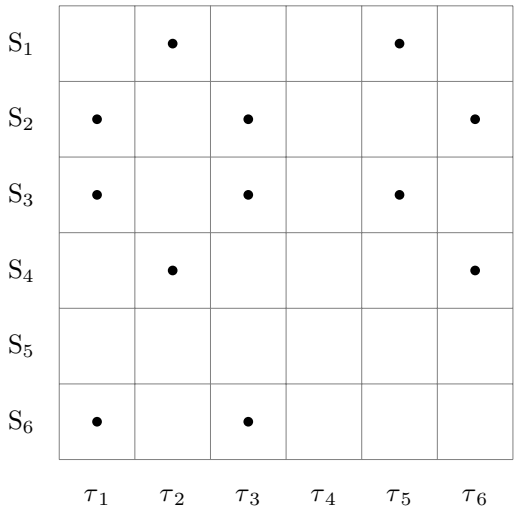
  $$\tau_1 : S_2; \tau_2 : S_1; \tau_3 : S_3; \tau_4 : \text{(none)}; \tau_5 : \text{(none)}; \tau_6 : S_1$$

  Then the sequence contributed by this u-v pair is $S_2S_1S_3{**}S_1$. '*'will be used when no edge is present in a particular quanta.
  Using this construction scheme, we get n*m sequences, each of length l, where n, m are the number of entities in both classes and l is the total number of quanta formed in the data log. These sequences form the sequence database D.
- Apply the sequence mining algorithm on the constructed D. Let the set of sequential patterns so obtained be SP. The claim is that set of continuous TAPs, hereafter called cTAP , form a subset of SP.

  $$\text{cTAP} \subseteq \text{SP}$$

This follows from the definition itself. Any frequent substring(i.e a TAP) is also a frequent subsequence, hence the sequence mining process will find it.

- Next we need to find a method which will discard the sequences in SP \ cTAP. Basically we need to check whether a given sequence $S \in SP$ occurs in at least α sequences in the database and starts in the same quantum in all of them.(Here α is the minimum support requirement).
- This can be done in following way: For each $Sq \in SP$, find the points of occurrence of substring Sq in S_{i} for $S_{\mathrm{i}} \in D$ and mark them as in the figure below(The dots show the position where Sq starts). This can be easily done through a pattern matching algorithm such as KMP.



If Sq is a valid TAP, then it must be true that for some quantum τ_{j}, it must start at τ_{j} in at least α sequences. Thus for each τ_{j}, we can simply count how many dots we encounter in that column(refer the figure above). If for some τ_{j}, the number of dots is more than α, then Sq is a valid TAP starting in quantum j. Else it is not a TAP and can be discarded.

For the purpose of this paper, let's call the above designed algorithm as *SeqMiner*

4 Experiments

4.1 Implementation

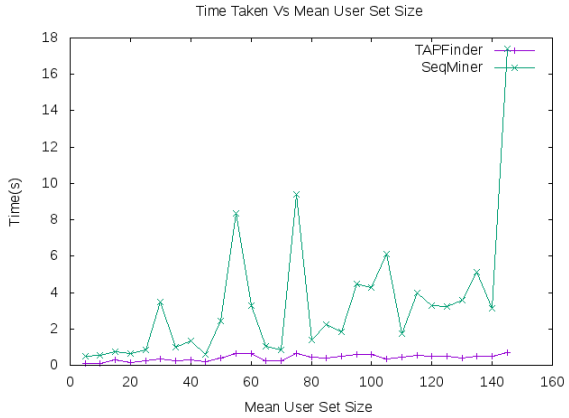
SeqMiner algorithm was implemented in C++11. Its data input and output formats are similar to *TAPFinder*. It internally calls the routine for *PrefixSpan* implementation of Java open source data mining library(SPMF: `http://www.philippe-fournier-viger.com/spmf/`). The step involving pattern matching is done by brute force, instead of KMP, as mentioned above.

4.2 Results

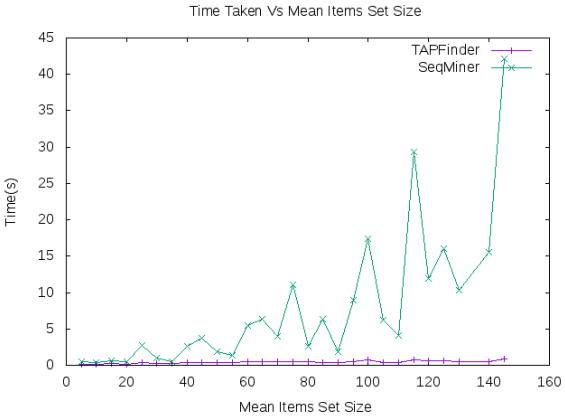
The experiments are done by running the *TAPfinder* algorithm and the *SeqMiner* algorithm on synthetic data. There are 4 parameters on which the complexity of the synthetic data depends:

- Mean Size of User Subset (30)
- Mean Size of Item Subset (20)
- Quantum Size (50)
- Quanta Length (10)

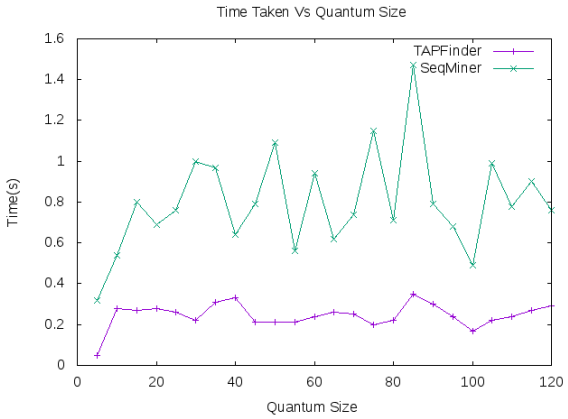
The values in brackets above are the default values. For each of the following experiments, one of the parameter is changed, keeping the others at their default values.



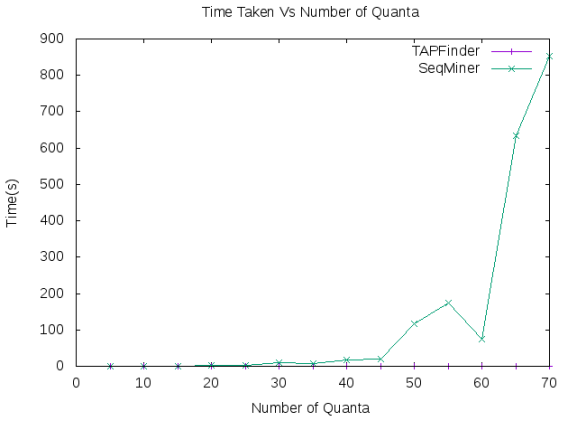
(b) Mean Users Set Size



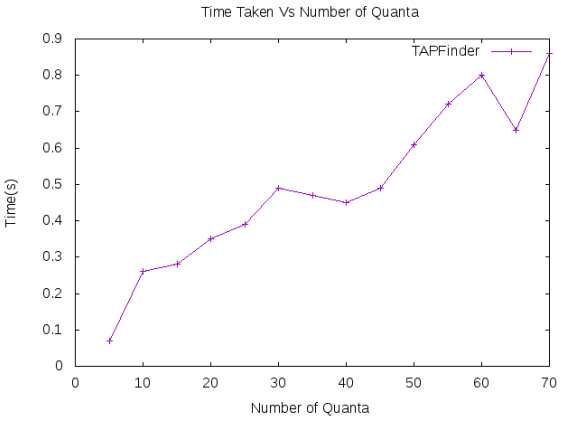
(c) Mean Items Set Size



(d) Quantum Size



(e) Number Of Quanta



(f) Number Of Quanta-TAPFinder

4.3 Observations

- From figure (a) and (b), the relationship between run-time and set size is almost linear for *TAPFinder*.This can be explained by the fact that *TAPFinder* works with limited number of pattern buckets at a time. The maximum number of such buckets at any quantum is

$$\frac{E_{max}}{\alpha} \propto \frac{Items \times Users}{\alpha}$$

. Thus increase in Item set or User set size linearly increases the processing time.

- Figures (a) and (b) also show that *SeqMiner* follows superlinear, probably exponential, relationship between run-time and set sizes. The *PrefixSpan* algorithm, which is the most time consuming part of *SeqMiner*, has exponential runtime complexity wrt the size of dataset. The number of sequences in SeqMining dataset is $Users \times Items$, thus increasing Item set or User set size would result in runtime increasing exponentially.

- There are a lot of oscillations in Figure (a) and (b). The reason for this is that the actual item and set size is different from the respective mean size. For a given set size, the actual size is a random variable, chosen from uniform distribution between 0 and $2 * \mu_{size}$. Uniform distribution enables any value in the given range to chosen with equal probability. Using a Gaussian distribution with low variance would result in actual set sizes to be closer to the mean value, giving smoother plots.

- From figure (c), both the algorithms are almost independent of quantum size. For the *SeqMiner* algorithm, the size of dataset is $|u-v \text{ pairs}| \times |quanta|$. The presence of an edge at u-v at some quantum 't' merely gives a label to the corresponding cell, which would have been a dummy value otherwise. Thus the runtime complexity is independent of quantum size. Similarly *TAPFinder* algorithm's runtime depends on $|quanta| \times buckets_{max}$ where $buckets_{max}$ depends on number of items, users and α as described above. This explains the independence of *TAPFinder* on quantum size.

- From figure (d) and (e), the time varies exponentially for *SeqMiner* wrt $|quanta|$ and linearly for *TAPFinder*, since $|$ Dataset Size $| = |Quanta| \times |u - vpairs|$ and *SeqMiner* runs exponentially in dataset size. *TAPFinder's* runtime depends on $|Quanta| \times buckets_{max}$, thus explaining linear relationship.

5 Future Work

- The above framework is valid for finding continuous simple TAPs through sequence mining. **Similar TAPs** are the set of sequences which are within fixed edit distance of each other and satisfy the constraints of min length and support. It remains to be seen if it is also possible to mine similar TAPs through sequence mining.

- To compare the performance of *SeqMiner* and *TAPFinder* on real life datasets.

References

[1] PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth
`http://idb.csie.ncku.edu.tw/tsengsm/COURSE/DM/Paper/PrefixSpan.pdf`

[2] Sequential PAttern Mining using A Bitmap Representation
`http://www.philippe-fournier-viger.com/spmf/SPAM.pdf`

[3] Fournier-Viger, P. and Gomariz, A. and Gueniche, T. and Soltani, A. and Wu., C. and Tseng, V. S. *SPMF: a Java Open-Source Pattern Mining Library*. Journal of Machine Learning Research (JMLR):
`http://www.philippe-fournier-viger.com/spmf/`