Temporal Cascade Detection

Agrim Gupta
Stanford University
agrim@stanford.edu

Abstract

Temporal networks are commonly used to represent systems where connections between elements are active only for restricted period of time, such as networks of telecommunication, neural signal processing, biochemical reactions and human social interactions. During the course of the project we develop a variety of algorithms to efficiently tackle the problem of detecting Top Cascades. Finally, we present a graph based approach which maps a sequence of events to a directed graph to detect Top Cascades.

1. Introduction

During the course of the project we tackled the problem of finding temporal cascades. The input consists of a set of events of the form Source (S), Destination (D), Start Time (T), Duration (δ). Two events are said to be W-adjacent, if the D in the first event is equal to S in the second event and the time difference between the end of the first event and the beginning of the second event is no longer than W. Further, two events are W-connected if there exists a sequence of events $e_i = e_{k_0} e_{k_1} \dots e_{k_n} = e_j$ such that all pairs of consecutive events are W-adjacent. Using these definitions we can define a temporal cascade as a set of events such that all pairs of events in it are W-connected.[1]

2. Methodology

In this section we describe the various approaches we developed for cascade detection.

2.1. Baseline Algorithm

We first started out with a naive approach. Given a set of events E, query event e_q and parameter W we use the algorithm described in Algorithm 1 to detect the forward cascade from this event. We initialize a queue by adding the query event. After, that we pop an element from the queue and find the events which are W-adjacent to this event by doing a linear search through the set of events E. The process is repeated till the queue is empty. To obtain all the cascades we can simply loop over all the events and perform Algorithm 1.

Data: $e_q \in E$, W
Result: Temporal Cascade starting from e_q
Add e_q to Queue;
while *Queue is not Empty* **do**
  e_c = Queue.pop;
  for $\forall e_i \in E$ **do**
    if e_i *is not visited and is W-adjacent to* e_c **then**
      Add e_i to Queue;
      Add e_i to Output;
    end
  end
end

Algorithm 1: Baseline Algorithm

Data: $e_q \in E$, W
Result: Temporal Cascade starting from e_q
Sort Events based on the Start time;
Add e_q to Queue;
while *Queue is not Empty* **do**
  e_c = Queue.pop;
  i = BinarySearch($T_{e_c} + \delta_{e_c}$);
  for $\forall e_j \in E \mid j >= i$ *and* $T_{e_j} - (T_{e_c} + \delta_{e_c}) \leq W$ **do**
    if e_j *is not visited and is W-adjacent to* e_c **then**
      Add e_j to Queue;
      Add e_j to Output;
    end
  end
end

Algorithm 2: Sorting Time-Based Algorithm

2.2. Sorting Based Approach

We can observe that to find W-adjacent event to a given event we go through the entire set of events in the baseline approach. This could be avoided if we sort the events based on either the source or the start time of the events. In both

Data: $e_q \in E$, W
Result: Temporal Cascade starting from e_q
Sort Events based on the Source;
Add e_q to Queue;
while *Queue is not Empty* **do**
  e_c = Queue.pop;
  i = BinarySearch(D_{e_c});
  for $\forall e_j \in E \mid j >= i$ *and* $S_{e_j} == D_{e_c}$ **do**
    if e_j *is not visited and is W-adjacent to* e_c **then**
      Add e_j to Queue;
      Add e_j to Output;
    end
  end
end

Algorithm 3: Sorting Source-Based Algorithm

the cases we would only need to examine a subset of events to determine if they are W-adjacent to a given event. Algorithm based on sorting with respect to Start Time and Source are described in Algorithm 2 and Algorithm 3 respectively.

2.3. Graph Based Approach

In Graph based approach we make use of the sorting based algorithm to map the set of input events to a directed graph. Each node in the graph represents an event and the an edge connects two W-adjacent events. Once the graph is generated we can find all the a cascade by doing a breadth first search to find all reachable nodes from a given query node (which represents an event). To find all the top cascades we will consider all the nodes with zero in-degree as query nodes.

2.4. Parallel Graph Based Approach

After the graph is generated all the Top cascades can be found by considering the zero in degree nodes in parallel. OpenMP was used to parallelize loops in the code for full utilization of our target multi-core platforms

3. Experimental Setup

All experiments were performed on a machine with 1TB RAM and 4x Intel CPU E7-4870 at 2.40GHz, each CPU has 10 cores and supports 20 hyperthreads for a total of 80 hyperthreads, running CentOS 6.4. Table 1 shows the datasets which were used to benchmark the algorithms developed.

4. Results And Discussion

Tables 2,3,4 show the benchmark results for detecting top cascades on different datasets for Sorting Based approach (Sort by Start Time), Graph Based Algorithm and Parallel Graph Based Approach. For Graph based algorithm

Dataset	Events	Events Per Source Node	Events Per Hour
Reality	45,480	561	**3.96**
FbMessages	59,835	44	**12.87**
SMS-A	548,182	**21**	67.51
YemenDataSet	161,306,628	**30**	216810.06

Table 1. Different Datasets Used for Benchmarking

W	Top Cascades	Baseline	Sort	Graph	Parallel
5	7	32.33	0.13	0.28	0.048
500	397	32.3	0.14	0.30	0.05
5000	2518	34.22	0.15	0.30	0.05
50000	6535	89.74	0.45	0.35	0.053

Table 2. Running Time of Different Algorithms for Reality Dataset

W	Top Cascades	Baseline	Sort	Graph	Parallel
5	556	45.3	0.18	0.28	0.06
500	61244	111.71	0.22	0.28	0.06
5000	339560	577.79	1.86	0.46	0.13
50000	25320485	52523.77	550.74	11.54	5.64

Table 3. Running Time of Different Algorithms for FbMessages Dataset

the time shown is cumulative of graph generation and cascade detection. We can see that graph based approach is generally faster than the sort based approach. Also, it gives the added flexibility of using other graph based algorithms to derive further insights from the data. As expected we get significant reductions in execution time by parallizing the algorithm. Table 5 shows the execution time to detect a single cascade in YemenDataset which has 7172 events. To estimate time for faster algorithms like Graph Based and Parallel Algorithm we ran the algorithm for $10,000$ iterations to estimate the running time of a single run. Interestingly, for this dataset we find that sorting by source gives significant improvement over the running time as compared to the sorting by start time. This motivated us to try both of these algorithms for graph generation. Table 6 show the time taken to generate graph using the two sort approaches. For Yemen dataset $W = 7200$ and for the rest datasets $W = 5000$. We can see that depending on the dataset either of the methods can perform better. On further analysis we observed that Yemen dataset has on average 30 events per source node which is significantly less than the number of events per hour which makes sorting based on source node perform better for graph generation.

W	Top Cascades	Baseline	Sort	Graph	Parallel
5	1347	3740.15	0.90	1.62	0.56
500	65114	4825.93	1.34	1.73	0.49
5000	61502	5933.83	5.05	2.83	0.44
50000	53415	8898.32	45.40	12.34	0.43

Table 4. Running Time of Different Algorithms for SMS-A Dataset

Method	Time
Baseline	97289.30
Sort-Time	56.68
Sort-Source	0.24
Graph	0.0026
Parallel	0.0016

Table 5. Running Time for Single Cascade Detection for Different Algorithms-Yemen Dataset

DataSet	Sort Source	Sort Time
Reality	0.16	**0.04**
FbMessages	0.19	**0.16**
SMS-A	**1.58**	1.86
YemenDataset	**825.94**	380160

Table 6. Time for Graph Generation

Characteristic	Value
Time Graph Gen	825.94
Time Cascade Detection (Graph)	29,720.86
Total Time	30,546.80
Top Cascades	27,740,352
Average Events Per Cascade	244.81
Max Cascade	169,468

Table 7. Cascade Detection Analysis - Yemen Dataset

5. Conclusion And Future Work

We were able to develop efficient algorithms for cascade detection. Currently, we can easily detect cascades for datasets having about hundreds of million events making datasets like Yemen datasets tractable for cascade detection. We would like to further investigate if we can optimize the parallel algorithm to lower the execution times even further.

References

[1] Kovanen, Lauri, et al. "Temporal motifs in time-dependent networks." Journal of Statistical Mechanics: Theory and Experiment 2011.11 (2011): P11005.