On Clustering in Temporal Networks

Peter Fagan
University College Cork

AM4090 written report
School of Mathematical Sciences, University College Cork, Ireland

Supervised by: Dr Philipp Hoevel

February 2019

On Clustering in Temporal Networks

Peter Fagan

School of Mathematical Sciences, University College Cork, Ireland.

February 2019

Abstract
The concept of clustering is well established in network science and useful to quantify the local cohesiveness of a network [15]. The local clustering coefficient, for instance, can be computed as the fraction of existing links between neighbours of a node and the maximum size of its neighbourhood [10]. Among other phenomena, this - together with a shortest average path length - explains the notion of small-world networks. Many networks, however, undergo temporal changes, which modify their structure. Then, the neighbourhood is also subject to change and the temporal ordering of link appearance becomes important to preserve causality. In this project, we aim to extend the concept of clustering to time-varying networks and apply it to both empirical and generic networks.
In this report we introduce the concept of a temporal network [6]. We extend the concept of network clustering and path lengths from the static case to the temporal case. Using these measures we implement heuristic algorithms and apply these algorithms to a real-world network [14]. In addition we highlight the drawbacks of static networks through identifying the number of non-causal paths in such networks [7]. Through varying time aggregations we will optimise the time frame over which we analyse a temporal network. Furthermore, we will extend the concept of a small-world network [10] through simulating the Watts-Strogatz model for temporal networks.

1. Introduction
Over the past number of years the use of complex networks has grown rapidly. Today we employ complex networks to solve complex tasks in areas such as epidemiology, genomics, transport, cybersecurity and sociology [2,3]. As methods for solving complex real-world problems become more data-centric, the analysis of complex networks emerges as a key method. Therefore, it is crucial that we develop a deep understanding of this growing field of research. Many real-world systems have been modelled using a static network representation. While a static representation of a network is extremely useful and insightful it suffers from assumptions that can lead to errors such as overestimating of spreading dynamics across the network [4]. As we strive to model real-world systems more accurately using complex networks, it has become evident that we must extend our definitions to include the additional dimension of time in our analysis. With the addition of time in our analysis we inherit many new challenges that were once trivial in the static case [6]. We must also be acutely aware of the dependence of our analysis of complex networks on modern technology and compute power. This dependence provides its own set of challenges which need to be accounted for.
In this report we will begin to discuss and tackle some of the challenges posed by including the additional dimension of time in our analysis. We will introduce the concept of a temporal network. From this foundational knowledge we will build our methods of analysis for temporal networks. Then we will apply these methods to real-world data using algorithms and derive insights from our analysis.

2. Temporal Networks
We define a graph $G = (V, E)$ as a set of objects, $V = \{v_1, v_2, v_3, \dots, v_N\}$, with a set of relationships, $E = \{e_{11}, e_{12}, e_{13}, \dots, e_{nn}\}$, existing between such objects. This definition of a graph is synonymous with the definition of a network except that our objects are called nodes and relationships between such objects are called links in the case of a network [5]. Both terms leave no restriction on what defines an object and/or a relationship between such objects. We can use the definition of a static graph to construct its temporal counterpart. The temporal definition remains the same as the static definition except the set E of edges includes an explicit dependence on time. In this way we define a temporal graph as $G = (V, E(t))$ where we denote, $E(t) = \{\tilde{e_{11}}, \tilde{e_{12}}, \tilde{e_{13}}, \dots, \tilde{e_{nn}}\}$ with each temporal edge $\tilde{e_{ij}} = (v_i, v_j, t, \delta t)$. Here t denotes the time at which the link between vertices (v_i, v_j) first becomes active and δt denotes the duration of the given link [6]. Now that we have our definition of a temporal network we can begin to ask, how can we carry out analysis on such a network? In general we represent temporal networks as a sequence of static network

representations called snapshots [7]. An important concept is the *temporal resolution* that we generate our snapshots with. The temporal resolution of a network is the size of the window of time we within which we consider active links to be active at the same time. It is the measure of time precision at which we examine our network. A temporal network's finest resolution is inherited from the raw data the network is constructed with. For example, if interactions are counted by the second, we cannot have a finer temporal resolution than 1 second. In some cases it might be of value to analyse the network at coarser time resolutions by building sequences of static representations of the temporal network through aggregation. We could decide to aggregate interactions within intervals of length 30-seconds into separate snapshots. In doing so, we need to be aware of the loss of temporal information in such aggregations. The static representation of any network is essentially the coarsest time resolution that can be constructed given temporal data. Therefore in order to leverage the extra information provided by temporal data we only aggregate when links are too sparse or we wish to analyse a different time scale.

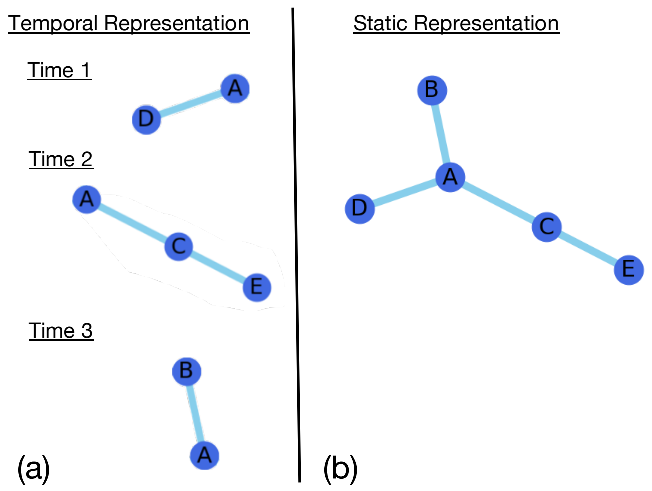


Fig 1: In panel (a) we observe the temporal network representation of a network as the sequence of snapshots along the time interval [1,3]. In panel (b) we have the static representation of the same network which can be got by aggregating links over the entire time interval [1,3].

An important property of temporal networks is the causality of paths. A *causal path* in a network is defined as a path between nodes that forms with respect to time [7]. Formally given a temporal network defined over some time interval $[0,T]$, $T \in \mathbb{Z}^+$ we can define a path by traversing the network with respect to time. To traverse multiple links in the network we require links to overlap. Two links,

$$\tilde{e_{ij}} = (v_i, v_j, t, \delta t)\,,\ \tilde{e_{jk}} = (v_j, v_k, \tau, \delta\tau)\,,$$

are said to overlap if one of the following conditions hold:

$$t \leq \tau \leq t + \delta t$$
$$\tau \leq t \leq \tau + \delta\tau. \qquad (1)$$

If two links overlap and share a node in common as above we can say that their nodes can be traversed. Using the notation above nodes $\{v_i, v_j, v_k\}$ can be traversed using links $\{\tilde{e_{ij}}, \tilde{e_{jk}}\}$ at times specified by the overlap. For directed networks in order to traverse overlapping links we have the extra requirement that the target node of the first link must be the source node of the second link. In this way a temporal network can be traversed. A *causal path* between nodes is defined as the consecutive sequence of nodes traversed in travelling between the starting and finishing node while obeying the conditions defined above. Looking at Figure 1 the path $D \rightarrow A \rightarrow C$ is a causal path, however, the path $C \rightarrow A \rightarrow D$ is not a causal path since it doesn't form with respect to time. It is important to note that in general we allow a single link to be traversed per time step. Hence, for example the path $A \rightarrow C \rightarrow E$ cannot be traversed in the interval $[2,3]$ since only a single link can be traversed in this time. It is also possible to specify the time taken to traverse specific links by weighting our network. Given this understanding of causal paths we can critique a static representation of a network. In static representations of real-world networks there are many non-causal paths. These non-causal paths introduce errors which often lead to an over estimation of spreading dynamics [4]. This motivates the temporal representation of a network.

In order to investigate how many non-causal paths are present in a static representation of a temporal network we can use a method of *unfolding accessibility* to calculate a networks *causal fidelity* [7]. This analysis can be helpful in developing a macroscopic view of a temporal network. First we consider the method of unfolding accessibility. Given a temporal representation of a networks in terms of a sequence of adjacency matrices $\{A_n\}$ we wish to examine how many causal paths are present in a network. We can do this by calculating accessibility matrices of our temporal network. First we will revise how to calculate such accessibility matrices in the static case and then extend this to the temporal case. In the static case we calculate the accessibility matrix for step n using Boolean matrix algebra as follows,

$$P_n = \bigvee_{i=1}^{n} A^i\ , \qquad (2)$$

At each step n our resulting matrix P_n at position (i,j) equals 1 if there exists a path between the nodes $\{v_i, v_j\}$ of length at most n and zero otherwise. Our Accessibility matrix saturates when $n = D$ where D is the diameter of the static network. To calculate the accessibility matrices of a temporal network we apply Boolean matrix multiplication [8] as below to our sequence of adjacency matrices.

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \odot \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$$

Fig 2: Example of Boolean matrix multiplication.

In the temporal case we note that only those paths which stem from active nodes in our first snapshot will be considered. This causes our accessibility matrix to underestimate network accessibility since causal paths starting at some future time point are essentially ignored. This is one of the drawbacks of methods involving matrix multiplication. We can remedy this issue by inducing memory in the network. This can be achieved by allowing for the inclusion of self-loops. We do this by adding the identity matrix to each adjacency matrix. This construct assumes that the network has infinite memory as seen in the paper [7]. We can calculate the accessibility matrix at step n as follows,

$$\mathcal{P}_n = \bigwedge_{i=1}^{n} (I + A)^i. \qquad (3)$$

In this report we allow for the modification of the above formulation by introducing finite memory to the network. This is achieved through a heuristic search of adjacency matrices for links and the creation of self-loops up to a specified memory length for locations where links are found. Implementation of this memory inclusion can be found in the code in the appendix of this report. The algorithm for carrying out such a search is as follows,

Algorithm (Memory Inclusion)

1. Store all snapshots as a sequence of adjacency matrices. Store another copy of the sequence which will be modified by adding memory terms call this the added memory sequence.

2. Initialise a search of the first adjacency matrix in the sequence for non-zero terms. If a non-zero term is found in position (i,j) then set entry (j,j)=1 for the next memory length adjacency matrices in the added memory sequence. Otherwise continue.

3. Once the search of the first adjacency matrix is complete iterate through the entire sequence of adjacency matrices in the original sequence according to the above procedure.

4. Return the added memory sequence.

We can define a measure of how well a static network represents a temporal network called causal fidelity denoted c [7]. We do so by calculating the ratio of the number of paths present in the saturated accessibility matrix of the temporal representation of a network when compared with the saturated accessibility matrix of the static representation. A convenient way to do so is by taking the density of our saturated accessibility matrices in each case.

$$\rho(A) = \frac{nnz(A)}{N^2}$$
$$c = \frac{\rho(\mathcal{P}_T)}{\rho(P_D)} \qquad (4)$$

Where D is the diameter of the static network and T is the maximal time in the temporal network and $nnz(A)$ denotes the number of non-zero elements in the adjacency matrix A . This calculation of causal fidelity is equivalent to taking the ratio of the number of causal paths in the static network all over all possible paths in the static network. Note that we may also plot the cumulative probability distribution of shortest paths present in the network using the accessibility matrices calculated by noting that the probability to find a path of length $l \leq n$ between two randomly chosen nodes in a connected network is given by the equation:

$$F_n = P(l \leq n) = \rho(\mathcal{P}_n)\,. \qquad (5)$$

We will explore these concepts further in our analysis of a real-world dataset.

3. Temporal Clustering

Network clustering is one of the fundamental measures characterising a complex network. In static networks we can calculate the global clustering coefficient or transitivity of a network by counting the fraction of transitive triples that are present in the network [5] . This measure indicates the tendency of nodes in the network to cluster together. It does so by measuring the likelihood that an open triple becomes a transitive triple (triangle). In layman's terms, in a social network it indicates the likelihood that two of your friends are friends with each other. This statistic can be formally calculated as follows:

$$C_{global} = \frac{(tr({A_{ij}}^3))}{\sum_j ({A_{ij}}^2 - I)} \qquad (6)$$

Where A_{ij} corresponds to the network's adjacency matrix. We will modify formula (6) and apply it to temporal networks. In the case of temporal networks we need to modify our definitions of a transitive triple and open triple. A transitive triple in the temporal case becomes a *temporal triangle*. A temporal triangle is a reciprocal causal path of topological length 3 between three unique nodes [9].

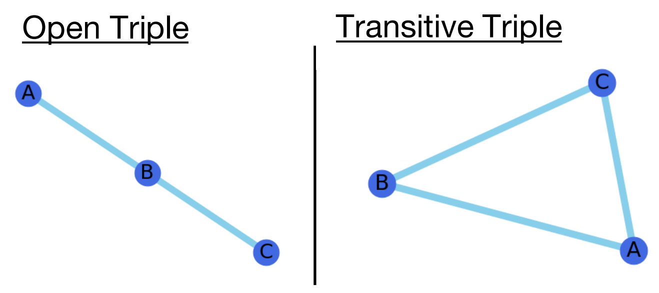


Fig 3: Examples of an open triple (left) and a transitive triple (right).

Taking the previously defined conditions for a causal path we simple need to include reciprocity and a constraint on the topological path length to get a temporal triangle. A more straight intuitive explanation is given in Figure 4.

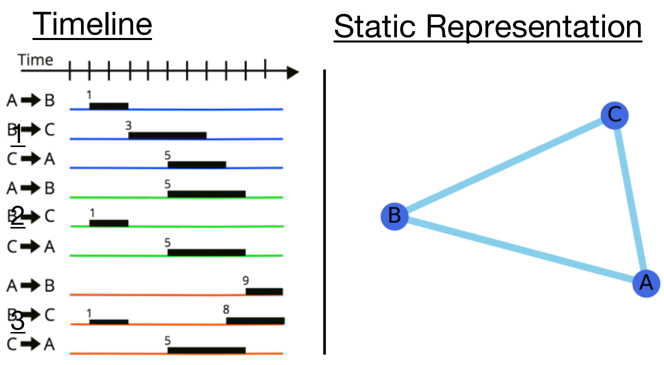


Fig 4: Timeline of links and link duration for three different cases (left). Case 1 illustrates the formation of a temporal triangle since overlapping links are causal, reciprocal and of length 3. Case 2 violates the causality condition as links do not overlap. Case 3 violates the reciprocity condition(also causality in the case of directed network) and hence does not form a temporal triangle.

An open triple in the temporal case is simply a causal path of length 2 between three unique nodes. In order to calculate the temporal clustering of a graph we must count the number of temporal triangles found in a temporal network. In order to do this we must implement an algorithm. The algorithm I created is defined as follows:

<u>Algorithm (Temporal Clustering)</u>

1. Initialise temporal_triples and temporal_triangles= 0. Initialise memory values for nodes. Store all snapshots as a sequence of adjacency matrices.

2. Initialise primary search for active links (i.e. non-zero entries) starting at the first adjacency matrix in the sequence. If a non-zero entry is found in (i,j) position initialise a secondary search of the next memory length consecutive adjacency matrices in the sequence at the jth row. Otherwise Continue.

3. For secondary search of adjacency matrices if a link is found in the (j,k) position increment the count of temporal_triples by 1 and terminate the secondary search of future adjacency matrix for this actor. Initialise a tertiary search for the next memory length consecutive adjacency matrices in the sequence at position (k,i). Otherwise continue.

4. For tertiary search of adjacency matrices if a link is found in position (k,i) increment the count of temporal_triangles by 1 and terminate the tertiary search for this actor. Otherwise continue.

5. Repeat this procedure starting the primary search at the next adjacency matrix in the sequence.

6.Return the ratio of triples to triangles for the entire network.

***Note: Primary search for adjacency matrices at the end of the sequence is significantly shortened. When appropriate, extend search by looping the memory searches to the start of the sequence of adjacency matrices.*

4. Temporal Paths

For temporal networks there are two types of paths. A *topological path* is defined as the number of nodes traversed in travelling from node v_i to node v_j. A *temporal path* is defined as the time taken in travelling from node v_i to node v_j [6]. If we are looking to find the shortest path from node v_i to node v_j we may consider a number of definitions including: the shortest temporal path, shortest topological path, length of the topological path corresponding to the shortest temporal path and length of the temporal path corresponding to the shortest topological path. As an example of how topological and temporal lengths may differ consider the timeline displayed in Figure 5. Each variant of paths have their own significance depending on the application however, for most analysis on temporal networks the shortest temporal path is sufficient. Hence, from here on when we mention shortest paths we refer to the shortest temporal path unless explicitly stated otherwise. To calculate the average shortest temporal path L we must compute the following [6]:

$$L = \frac{1}{N(N-1)} \sum_{i,j} d_{i,j} , \qquad (7)$$

where $d_{i,j}$ is the length of the shortest temporal path between nodes (v_i, v_j) and N denotes the number of nodes in the network.

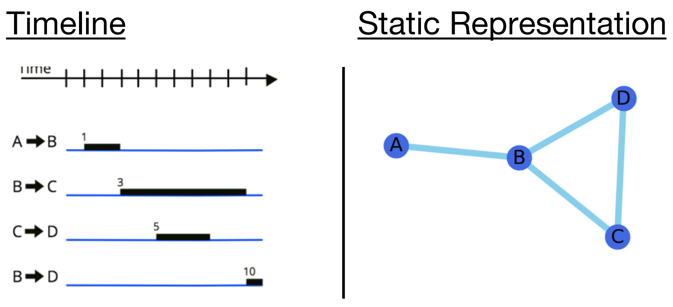


Fig 5: Timeline of links and link durations (left). Static Representation of the network (right). Assuming instantaneous length of time to traverse a link the above timeline represents the difference in topological and temporal path lengths. Consider the paths between A and D. The path with the shortest topological length is clearly A→B→D of topological length 3. However, the temporal length of this path is 7. The path A→B→C→D is of topological length 4 which is longer than that of the first path we considered however, it's temporal length is just 4. We could also consider various combinations such as the topological length of the shortest temporal path. In this case that would be 4 for the paths between A and D.

In analysing temporal networks it can be the case that many of these node pairs will not have any causal path and hence $d_{i,j} = \infty$. This can cause our calculations to of L to blow-up. To account for this we can take two approaches. We can assess the network efficiency as defined in [6]:

$$E = \frac{1}{N(N-1)} \sum_{i,j} \frac{1}{d_{i,j}} . \qquad (8)$$

This method will allow those node pairs with infinite temporal distance to go to zero. Alternatively we can ignore these terms where $d_{i,j} = \infty$ and alter the normalisation factor to only account of pairs where there exists a path of finite length.

$$\tilde{L} = \frac{1}{N^*(N^*-1)} \sum_{i,j} d_{i,j} \qquad (9)$$

$$N^* = \#active\ paths,$$

In order to calculate the shortest temporal paths we will use a modified version of unfolding accessibility. The algorithm I created is as follows:

<u>Algorithm (Shortest Temporal Path)</u>

1. Store all snapshots as a sequence of adjacency matrices.

2. Extend the memory of nodes by increasing the duration of links by adding unit terms to memory length consecutive adjacency matrices following an active link.

3. Choose the first adjacency matrix in the sequence as our starting point. Store this first adjacency matrix as our first accessibility matrix. Perform matrix multiplication on this adjacency matrix by the next adjacency matrix in the sequence. Store the resulting accessibility matrix.

4. Multiply the resulting accessibility matrix from the previous step by the next adjacency matrix in the original sequence and continue this iterative process until the last adjacency matrix is reached.

5. Search the sequence of generated accessibility matrices one at a time for non-zero terms. When a non-zero term is found at position (i,j) for the first time, store the position of the accessibility matrix in the sequence as it's temporal path length.

6. Recalculate the accessibility matrices for a new starting point and repeat procedure.

7. Find the shortest path length for a given position from all starting points.

8. Compute the average temporal path length of the network

5. Temporal Small-World

The notion of a small-world network was first published in the seminal paper [10] by Duncan Watts and Steven Strogatz. The small-world property is commonly used to classify a network. Many studies have investigated dynamics on small-world networks [11]. In this report we will attempt to extend this definition to temporal networks in light of our previous work on clustering and path lengths. The aim of classifying temporal small-world networks is to discount those networks that display large amount of clustering and short average path length due to non-causal paths rather than causal paths. We argue that these networks in the static case should not be classified as small-world networks due to their dependence on paths which do not form with respect to time. To classify a small-world network we will use the measures and algorithms we have previously defined. Those readers familiar with [10] may have

already noticed that the path length considered in the study were topological path lengths and not temporal path lengths(since it was a static network). Furthermore, the calculation of the clustering coefficient is based on a different formula to the formula we extended to the temporal case. We will consider other methods for analysing the small-worldness of a network in the discussion section.
In order to investigate small-world properties on temporal networks, we generate a sequence of static graphs using the Watts-Strogatz model [10]. To construct a static network according to the Watts-Strogatz model we do as follows:

Watts-Strogatz Model (Construction)
1. Construct a k regular ring lattice with N nodes, having each node in the network connected to k/2 neighbours on either side.

2. For each node n_i in the network consider each link connecting the node n_i to each one of its neighbours. Rewire each link randomly to another node with the probability α.

***Note: If a node is being rewired at random to another node, it can only be rewired in such a way as to avoid creating self loops or link duplication.*

As our probability for rewiring links approaches 1 our static network becomes more random. For certain probabilities between 0 and 1 our network is a small-world network. This is the case when the relative clustering coefficient remains high and the relative shortest path lengths remain low.

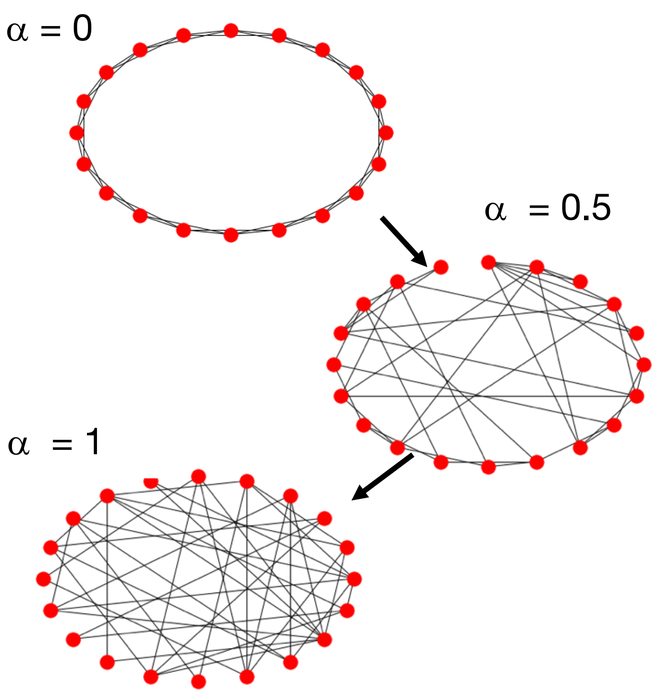


Fig 6: Illustration of Watts-Strogatz networks for varying rewiring probability α.

Given a sequence of static networks we transform these networks to temporal graphs by modelling link frequencies over a predefined set of windows. One can assume an underlying distribution for the activation frequency of links for example the binomial. To simplify our analysis we assume that all links are active for an equal frequency and that these activations are distributed randomly amongst snapshots for each link. Note that this is an extremely simplified model it is possible to build significantly upon this simulation model. This model does not assume any inter/intra snapshot relationships.

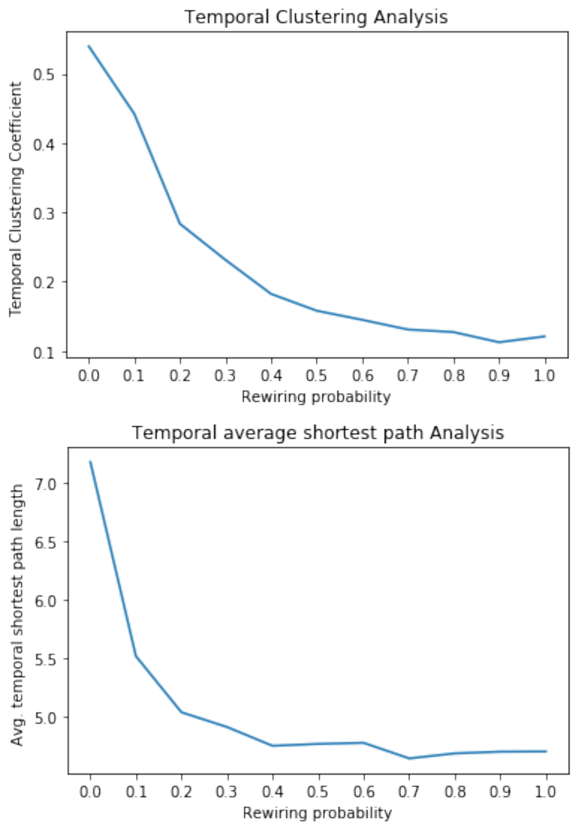
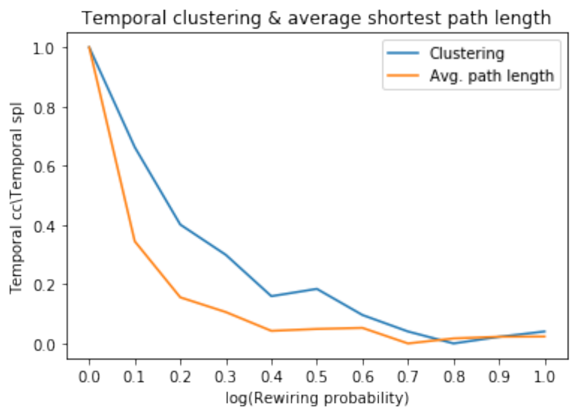


Fig 7: (Top) plot of temporal clustering coefficient vs rewiring probability



α of static network from which we generate our temporal network. (Middle) plot of average shortest temporal path length vs rewiring probability α of static network from which we generate our temporal network.(Bottom) both plots normalised and plotted together.

Having generated the temporal representations of these networks over varying rewiring probabilities we can apply our algorithms to see how both the clustering and average shortest temporal path vary. From Figure 7 we can clearly see that both the clustering and the temporal path lengths depend upon the static network topology as we see they both decrease as we vary the probability of rewiring links in the Watts-Strogatz model. It is important to note that this is for the case where links are modelled with a fixed frequency. In adopting a more sophisticated model we may encounter different results. The importance of the static network topology may become less significant when compared to the distribution of links across time.

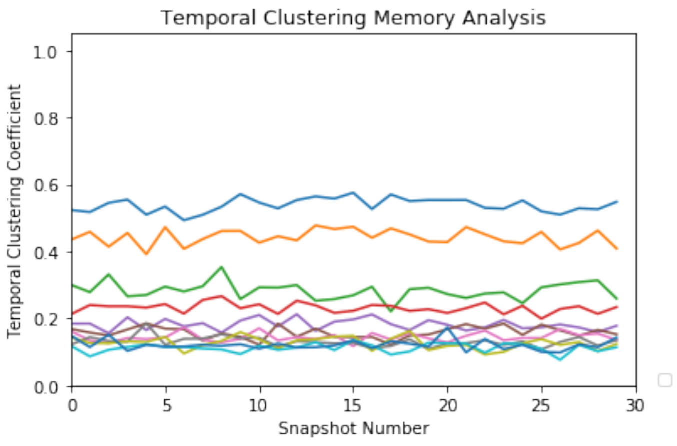


Fig 8: Plot of the changes in clustering over snapshots for different rewiring probabilities. Each line represents a full iteration for given rewiring probability.

As we can see for each iteration in Figure 8 the clustering remains almost perfectly constant this is due to assuming the constant frequency of links and distributing them randomly between snapshots. In future reports on the topic of temporal networks and in-depth exploration of these types of simulations could be of great value and insight. For our purposes we have shown that both temporal clustering and average temporal path length depend on the static network topology and that they decrease as the static network topology becomes more random. An approach similar to the one found in [12] is also worth investigating. In this study they calculate the temporal clustering coefficient by calculating the temporal correlation coefficient:

$$C_i = \frac{1}{T-1}\sum_{t=1}^{T-1}\frac{\sum_j a_{ij}(t)a_{ij}(t+1)}{\sqrt{[\sum_j a_{ij}(t)][\sum_j a_{ij}(t+1)]}}$$

$$C = \frac{\sum_i C_i}{N} \tag{10}$$

The measure for the shortest temporal path length is the same as we have defined in (7).

6. Analysis & Results

The dataset analysed in this report is from a set of face-to-face interactions recorded over a three day academic conference. The dataset is made publically available online at the following source [14]. This data was constructed using RFID sensors which detect when two participants of the conference were engaged in a face-to-face interaction. The original temporal resolution of the data is 20 seconds (snapshot sizes of 20 seconds). In this form the data is too sparse. Hence, the dataset is aggregated to the more suitable window of one hour. In this form we had 59 snapshots to analyse (the conference begins early on the first day and concludes early on the third day).

To begin analysing this dataset the causal fidelity was calculated to gauge how well the static network represents the temporal network. In this case we found the static fidelity to be $c \approx 0.79$. Therefore, approximately 21% of paths in the static representation are non-causal paths. This motivates our analysis of the network as a temporal network. Therefore we proceed with our analysis by evaluating the clustering of the network overtime. We will also alter the memory parameter which will allow for temporal triangles of a longer duration to form. In this way we can understand the importance of memory in the network by examining the rate of increase in our clustering coefficient. As we can see from Figure 9, that the clustering increases with increasing memory as we would expect. Furthermore, our algorithm correctly indicates the lack of clustering at inactive periods such as during the night when the conference has adjourned. conference may provide insights into triggers for increased socialisation/clustering.

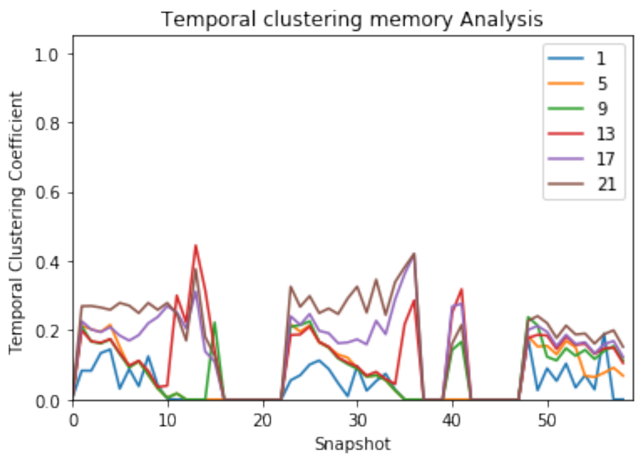


Fig 9: Plot of temporal clustering coefficient for varying values of memory over the range of snapshots for the dataset (legend).

We can also see particular peaks in clustering which indicate times at which the network clustering is most active. Identifying these peaks and coupling them with supplementary information about the conference has the potential to uncover further insights. Plotting our average clustering coefficient for increasing memory shows a general increasing trend as shown in Figure 10.

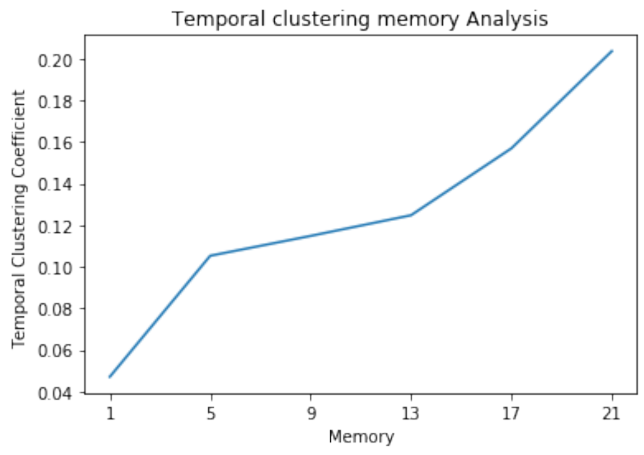


Fig 10: Plot of temporal clustering coefficient for increasing values of memory.

7. Discussion

The main benefit of analysing temporal networks is the exclusion of non-causal path and the inclusion of analysis over specific time intervals. From the above analysis it is clear that the clustering and temporal path lengths of a temporal network depend on both the static structure of the network as well as the frequency and durations of link appearances. Further investigation into how significant each of these factors is to the analysis of temporal networks is warranted to help understand the properties of temporal networks. In this report, we choose to model the network clustering using a heuristic approach of counting temporal triangles. It should be noted that other variants exist and can be formulated. Another study [12] used the temporal correlation coefficient as a measure of temporal network clustering. This method is based upon the idea that persistent edges form clusters in a temporal network. Edge persistence can be related to the frequency of link activations. I hypothesise, that temporal clusters are formed by a combination of static topological structure, node/link memory and link persistence. In future investigations I would try to evaluate the relationship between these factors and how a network tends to cluster. Furthermore, a thorough examination of the algorithms used and the efficiency of such algorithms is an important challenge that needs to be confronted. In this application it was necessary to rent a virtual machine in order to improve computation speed. Other approaches such as using Numba to optimise the python compiler as well as parallelisation were considered. The topic of algorithmic efficiency is a report in and of itself which deserves more attention in future research. An area also worth exploring some more is that of activity-driven networks which aim to tackle the problem of temporal networks based on the activity of links [13].

8. Conclusions

In this report I have introduced an approach that extends the concept of clustering to temporal networks. I have also modified the method of unfolding accessibility as described in [7] and formulated a finite memory version. I modified the method of unfolding accessibility and used it to calculate shortest temporal paths. I simulated temporal small-world networks using a novel approach with a simplistic model of link activity. Finally, I applied some of these methods to a real-world dataset as a proof of concept and to gain insights into the dataset.

9. Acknowledgements

I would like to thank Dr Philipp Hoevel for his help and guidance in formulating this report.

10. References

[1] Dai X, Hu M, Tian W, Xie D, Hu B (2016). Application of Epidemiology Model on Complex Networks in Propagation Dynamics of Airspace Congestion. PLoS ONE 11(6): e0157945.

[2] Lentz HHK, Koher A, Hövel P, Gethmann J, Sauter-Louis C, Selhorst T, et al. (2016). Disease Spread through Animal Movements: A Static and Temporal Network Analysis of Pig Trade in Germany. PLoS ONE 11(5): e0155196.

[3] Crawford J, Milenković T (2018). ClueNet: Clustering a temporal network based on topological similarity rather than denseness. PLoS ONE 13(5): e0195993.

[4] Koher A, Lentz HHK, Hövel P, Sokolov IM (2016). Infections on Temporal Networks—A Matrix-Based Approach. PLoS ONE 11(4): e0151209.

[5] Albert Laszlo Barabasi (2016). Network Science, Cambridge University Press.

[6] Nicosia Vincenzo, John Tang, Cecilia Mascolo, Mirco Musolesi, Giovanni Russo, & Vito Latora (2013). Graph Metrics for Temporal Networks. Temporal Networks, Understanding Complex Systems. Springer-Verlag Berlin Heidelberg, p. 15.

[7] H. K. Lentz, Thomas Selhorst, and Igor M. Sokolov (2012). Unfolding Accessibility Provides a Macroscopic Approach to Temporal Networks. Phys. Rev. Lett. 110, 118701.

[8] Fischer, M.J., & Meyer, A.R. (1971). Boolean Matrix Multiplication and Transitive Closure. 12th Annual Symposium on Switching and Automata Theory (swat 1971), East Lansing, MI, USA, 1971, pp. 129-131.

[9] Cui, Jing & Zhang, Yi-Qing & Li, Xiang (2013).On the clustering coefficients of temporal networks and epidemic dynamics. Proceedings - IEEE International Symposium on Circuits and Systems. 2299-2302.

[10] D. J. Watts and S. J. Strogatz (1998). Collective dynamics of "small-world" networks, Nature 393, 440.

[11] Alun L. Lloyd, Steve Valeika, and Ariel Cintr´on-Arias (2006). Infection dynamics on small-world networks. 10.1090/conm/410/07729.

[12] J. Tang, S. Scellato, M. Musolesi, C. Mascolo, V.Latora (2010). Small-world behavior in time-varying graph. Phys. Rev. E 81, 055101(R).

[13] Nicola Perra, Bruno Gon¸calves, Romualdo Pastor-Satorras , Alessandro Vespignani1 (2012). Activity driven modeling of time varying networks. Nature Scientific Reports 2, 469.

[14] Mathieu Génois and Alain Barrat (2018). Can co-location be used as a proxy for face-to-face contacts? EPJ Data Science2018 7:11.

[15] M. E. J. Newman (2001). Clustering and preferential attachment in growing networks, Phys. Rev. E 364, 025102(R).

11. Appendix

Portion of Jupyter notebook code used to generate plots and perform analysis. Full notebook accessible from GitHub repository at the following link: https://github.com/peterfaganucc/Temporal-Networks-Clustering.

```
#Implementation of my own algorithms completely individual work no external resources used.
#Please note that the following code needs improvements in stability and efficiency.
#Given more time I would formalise this code to be closer to industry standard as well as including extensive commenting.
#Note this would likely mean recoding completely in C or another more efficient programming language or
#using an interpreter (Numba).
# All precautions have been made to exclude any errors however,
#further unit tests and revision of code are warranted given time to ensure there are no underlying errors.

def to_numpy_matrix(adjacency_matrices,max_window, np_matrices):
    """ Converts AdjMatrixSequence function output to numpy matrices."""
    for i in range(0,max_window):
        np_matrices[i] = np.matrix(adjacency_matrices[i].toarray())

    nrows = ncols = len(np_matrices[0])



def np_mat_mul(matrix1,matrix2):
    """Multiplication of two symmetric numpy matrices"""
    nrows=ncols=len(matrix1)
    res = [[0 for x in range(nrows)] for y in range(ncols)]

    matrix1 = matrix1.tolist()
    matrix2 = matrix2.tolist()
    # explicit for loops
    for i in range(len(matrix1)):
        for j in range(len(matrix2[0])):
            for k in range(len(matrix2)):

                # resulted matrix
                res[i][j] += matrix1[i][k] * matrix2[k][j]

    return np.asmatrix(res)

def np_mat_mul_bool(matrix1,matrix2):
    """Boolean Multiplication of two symmetric numpy matrices"""
    nrows=ncols=len(matrix1)
    res = [[0 for x in range(nrows)] for y in range(ncols)]

    matrix1 = matrix1.tolist()
    matrix2 = matrix2.tolist()
    # explicit for loops
    for i in range(len(matrix1)):
        for j in range(len(matrix2)):
            for k in range(len(matrix2)):

                # resulted matrix
                if (matrix1[i][k] * matrix2[k][j] == 1):
                    res[i][j] = 1
                else:
                    continue

    return np.asmatrix(res)
```

```
def include_memory_link(np_matrices,memory):
    """Extends duration of a link to the defined memory. For example suppose that at time t=3 the is a link between nodes
i & j.
       Given memory term of 2 we add 1 to the i,j position in the adjacency matrices at time t=4,t=5.
       Current limitation, doesn't include memory to terms where search of snapshots exceeds snapshots window
    """
    nrows = ncols = len(np_matrices[0])
    np_matrices_mem = {}
    for t in range(len(np_matrices)):
        np_matrices_mem[t] = np.zeros(shape=(nrows,ncols))


    for j in range(len(np_matrices)-(memory+1)):
        for x in range(nrows):
             for y in range(ncols):
                if (np_matrices[j][x,y]==1):
                    for k in range(j,j+memory+1):
                        np_matrices_mem[k][x,y] = 1
                else:
                    continue

    for l in range(len(np_matrices)-(memory+1),len(np_matrices)):
        np_matrices_mem[l] = np_matrices[l]
    return np_matrices_mem


def include_memory_node(np_matrices,memory):
    """Extends duration of a link to the defined memory. For example suppose that at time t=3 the is a link between nodes
i & j.
       Given memory term of 2 we add 1 to the i,j position in the adjacency matrices at time t=4,t=5.
       Current limitation, doesn't include memory to terms where search of snapshots exceeds snapshots window
    """
    nrows = ncols = len(np_matrices[0])
    np_matrices_mem = {}
    for t in range(len(np_matrices)):
        np_matrices_mem[t] = np.zeros(shape=(nrows,ncols))


    for j in range(len(np_matrices)-(memory+1)):
        for x in range(nrows):
             for y in range(ncols):
                if (np_matrices[j][x,y]==1):
                    for k in range(j,j+memory+1):
                        np_matrices_mem[k][y,y] = 1
                else:
                    continue

    for l in range(len(np_matrices)):
        np_matrices_mem[l] = np_matrices[l] + np_matrices_mem[l]
    return np_matrices_mem




def make_loop(np_matrices):
    '''Takes a series of adjacency matrix snapshots and doubles the length looping around the same set of snapshots
twice'''
    snapshots = len(np_matrices)
    for j in range(snapshots):
        np_matrices[j+snapshots] = (np_matrices[j])
    return np_matrices
```

```
def temporal_network_clustering(np_matrices,memory):
    """Implementation of clustering algorithm from paper.
    Calculates the number of triangles and triples based on searching the network using adjacency matrices."""
    triples = 0
    triangles = 0
    nrows = ncols = len(np_matrices[0])
    snapshots = len(np_matrices)
    make_loop(np_matrices)
    np_matrices[-1] = np.zeros(shape=(nrows,ncols))
    if(snapshots) <= 0 :
        print('The memory length entered is too large for meaningful calculations')
    else:
        for t in range(0,snapshots):
            for i in range(nrows):
                for j in range(ncols):
                    if ((np_matrices[t][i,j] == 1) and (np_matrices[t-1][i,j] == 0)):
                        for k in range(t+1,t+1+memory):
                            for l in range(ncols):
                                if (np_matrices[k][j,l] == 1) and (k==t+1):
                                    triples+=1
                                    for m in range(k+1,k+1+memory):
                                        if  (np_matrices[m][l,i] == 1) and (m == k+1):
                                            triangles += 1
                                            break

                                        elif (np_matrices[m][l,i] == 1) and (np_matrices[m-1][l,i] == 0) and (m>k+1):
                                            triangles += 1
                                            break
                                        else:
                                            continue

                                elif (np_matrices[k][j,l] == 1) and (np_matrices[k-1][j,l]==0) and (k>t+1):
                                    triples +=1
                                    for m in range(k+1,k+1+memory):
                                        if  (np_matrices[m][l,i] == 1) and (m==k+1):
                                            triangles += 1
                                            break

                                        elif (np_matrices[m][l,i] == 1) and (np_matrices[m-1][l,i] == 0) and m>k+1:
                                            triangles += 1
                                            break
                                        else:
                                            continue

                                else:
                                    continue


                    else:
                        continue
        if (triples==0):
            return 0
        else:
            return(triangles/triples)
```

```
def temporal_network_clust_plot(np_matrices,memory):
    """Calculates the number of triangles and triples based on searching the network using adjacency matrices and plots
the results"""
    snapshots = len(np_matrices)
    triples = [0]*(snapshots)
    triangles = [0]*(snapshots)
    clustering_nums = [0]*(snapshots)
    nrows = ncols = len(np_matrices[0])
    make_loop(np_matrices)
    np_matrices[-1] = np.zeros(shape=(nrows,ncols))

    if(snapshots - (1.5*memory)) <= 0 :
        print('The memory length entered is too large for meaningful calculations')
    else:
        for t in range(snapshots):
            for i in range(nrows):
                for j in range(ncols):
                    if ((np_matrices[t][i,j] == 1) and (np_matrices[t-1][i,j] == 0)):
                        for k in range(t+1,t+1+memory):
                            for l in range(ncols):
                                if (np_matrices[k][j,l] == 1) and (k==t+1):
                                    triples[t] += 1
                                    for m in range(k+1,k+1+memory):
                                        if  (np_matrices[m][l,i] == 1) and (m == k+1):
                                            triangles[t] += 1
                                            break

                                        elif (np_matrices[m][l,i] == 1) and (np_matrices[m-1][l,i] == 0) and (m>k+1):
                                            triangles[t] += 1
                                            break
                                        else:
                                            continue

                                elif (np_matrices[k][j,l] == 1) and (np_matrices[k-1][j,l]==0) and (k>t+1):
                                    triples[t] +=1
                                    for m in range(k+1,k+1+memory):
                                        if  (np_matrices[m][l,i] == 1) and (m==k+1):
                                            triangles[t] += 1
                                            break

                                        elif (np_matrices[m][l,i] == 1) and (np_matrices[m-1][l,i] == 0) and m>k+1:
                                            triangles[t] += 1
                                            break
                                        else:
                                            continue

                                else:
                                    continue


                    else:
                        continue

        for l in range(len(clustering_nums)):
            if triples[l] == 0 :
                clustering_nums[l] = 0
            else:
                clustering_nums[l] = triangles[l]/triples[l]

        plt.ylim(0,1.05)
        plt.xlim(0,snapshots)
        plt.plot(clustering_nums)
```

```
#Algorithms for shortest temporal path needs to be improved, first definition take to long to run and needs revision
#Second defintion beneath works quicker however simply averages over each snapshot calculation.
def temporal_path_length(np_matrices,memory):
    '''Implementation of algorithm outlined in the paper. Calculate the shortest temporal path length utilising unfolding
accessibility'''

    paths_active=[]
    snapshots = len(np_matrices)
    N = nrows=ncols=len(np_matrices[0])
    np_matrices_mem = include_memory_node(np_matrices,memory)
    make_loop(np_matrices_mem)


    for t in range(snapshots):
        access={}
        access[-1] = np.zeros(shape=(nrows,ncols))
        access[0]= np_matrices_mem[t]
        for i in range(snapshots):
            res = np_mat_mul_bool(access[i],np_matrices_mem[i+1+t])
            access[i+1]=(res)


        times = [[0 for x in range(nrows)] for y in range(ncols)]

        for k in range(snapshots):
            for i in range(nrows):
                for j in range(ncols):
                    if (access[k][i,j] == 1) and (access[k-1][i,j]==0) and (times[i][j]==0):
                        times[i][j] = k+1

                    else:
                        continue




        start_nodes = len(np.unique(np.nonzero(access[0])[1]))
        active_nodes = len(np.unique(np.nonzero(times)[1]))
        if start_nodes == 0:
            paths_active.append([[0 for x in range(nrows)] for y in range(ncols)])
        else:
            paths_active.append(times)
    shortest_times = [[0 for x in range(nrows)] for y in range(ncols)]
    for m in range(nrows):
        for n in range(ncols):
            temp = []
            for p in range(len(paths_active)):
                temp.append(paths_active[p][m][n])
            shortest_times[m][n] = min(temp)


    active_nodes = len(np.unique(np.nonzero(shortest_times)[1]))
    return(np.sum(shortest_times)/(active_nodes*(active_nodes-1)))
```

```
def temporal_path_length(np_matrices,memory):
    '''Implementation of algorithm outlined in the paper. Calculate the shortest temporal path length utilising unfolding
accessibility'''

    paths_active=[]
    snapshots = len(np_matrices)
    N = nrows=ncols=len(np_matrices[0])
    np_matrices_mem = include_memory_node(np_matrices,memory)
    make_loop(np_matrices_mem)


    for t in range(snapshots):
        access={}
        access[-1] = np.zeros(shape=(nrows,ncols))
        access[0]= np_matrices_mem[t]
        for i in range(snapshots):
            res = np_mat_mul_bool(access[i],np_matrices_mem[i+1+t])
            access[i+1]=(res)


        times = [[0 for x in range(nrows)] for y in range(ncols)]

        for k in range(snapshots):
            for i in range(nrows):
                for j in range(ncols):
                    if (access[k][i,j] == 1) and (access[k-1][i,j]==0) and (times[i][j]==0):
                        times[i][j] = k+1

                    else:
                        continue




        start_nodes = len(np.unique(np.nonzero(access[0])[1]))
        active_links = len(np.unique(np.nonzero(times)[1]))
        if start_nodes == 0:
            paths_active.append(0)
        else:
            paths_active.append((np.sum(times))/(2*(active_links*(active_links-1))))

    return(np.average(paths_active))
```