A Survey on the Different Approaches to Temporal Node Centralities

Simone Boscolo*, Bruno Principe*, Raazia Tariq

University of Padua, Veneto, Italy

simone.boscolo.2@studenti.unipd.it , bruno.principe@studenti.unipd.it,

raazia.tariq@studenti.unipd.it

Abstract

In graph theory it is not unpopular to find applications in which it is asked to find the most central nodes in a network. However, differently from static networks from which there are different solutions from the literature , without reaching the scalability in the biggest datasets, the dynamic network instead has not studied much. Every now and then there were proposed variant methods or new ones, nonetheless no one reached scalability in terms of memory or time. In fact, it is a well-known NP-hard or polynomial in a very limited number of cases. Here we are proposing a survey of some of the solutions proposed until now, with implementations of un-released ones and re-implementation for some of them that used deprecated languages , methods and not standard manual preprocessing in dataset. Then we are going to show statistics and analysis of the different approaches.

Introduction

A static world does not exist, everything is fickle and is enduring the burden of time. This is a logic that works in most of the cases in practice that is a counter-walk to what the abstract world establishes, something that we can relate to in the mathematical fields but not in a physical world. With this premise we are going to dive into a sub-branch of the Mathematical Graph Theory more in details Temporal Graph Theory, herein we find applications to multiple sectors as: social networks, co-authorship networks, biology networks, ontology summarization, ad-hoc wireless or radio networks, mobile phone calls, epidemiology networks; just to mention some of them. The dynamicity of this type of networks makes possible the creation of multiple interactions among vertices that can be created or crossed out within a specific span of time, so by giving a general definition: let $G = (V, E)$ be an undirected static graph with *V* a finite set of nodes and *E* a finite set of edges for which$E \subseteq \{[u, v] \epsilon\, VxV, unordered\ pair\}$, and a directed graph *G = (V, E)* with a finite set of edges as $E \subseteq \{(u, v) \epsilon\, VxV, ordered\ pair\}$.

Meanwhile a temporal graph $Tg = (V, \xi, \delta)$ with *V* the same finite set of vertices, and $\boldsymbol{\varepsilon}$ the finite set of edges that this time is formed by a triplets $e = (n, v, t)$ where n is the source node, v the destination node and the time for which the transition takes place. $\delta \in \mathbb{N}$ is the delay for an edge to propagate its information to the destination [1]. In this case the analysis of centrality of a node is not only based on the single definition of shortest or restless path, but herein the concept key is the presence of multiple definitions of what are temporal walks and temporal paths. For *Temporal Walk* we define an ordered sequences of arcs $(e_1, \ldots, e_k) \in \xi^k$, the length of the walk W is |*W*|=*k*. More formally the edges are temporally sorted $e_i = (u, v, t_i) \preccurlyeq e_{i+1} = (v, w, t_{i+1})$, where $t_i \leq t_{i+1}$, where $i \epsilon [1, k]$. In the other hand a *Temporal Path* pass the same vertices just once [2], we can define it as P = $\langle e_1, \ldots\ldots, e_k \rangle$ in which, by recalling the same properties of walks, the edges are sorted in timespan and the path length is k. [3] Keep in mind that in the literature there is a misuse of these two words. They are used interchangeably despite how they describe two different concepts. Coming back to the theory, it is used to differentiate the types of walks as follow:

- *shortest walk* is the walk with the minimum total number of time edges crossed until now.
- *fastest walk* is the walk with the edges for which has the minimum difference from the first and the last timestamp.
- *restless walk* is a walk between two vertices for which the transition time to the last edges is the minimum.

One of the main points of the graph analysis is also the computation of the node importance in a network, or so-called Node Centrality. Its definition is not the only one but changes depending on the type of centrality adopted, and nowadays does not exist a method for the Temporal Graph that scales well both in time and in memory. In this paper we are presenting some of the methods discovered so far showing its theory, the changes in the implementation done for making possible the experiment and finally describing the numerical results and statistics between them.

Algorithms

ONBRA [3]

ONBRA is an approximation algorithm where the problem instead of being focused in the computation of the classical static betweenness centrality, that is defined as

$$b(v) = \frac{1}{n(n-1)} \sum_{s,z \in V, s \neq z} \frac{\sigma_{s,z}^{sh}(v)}{\sigma_{s,z}^{sh}},$$

with $\sigma_{s,z}^{sh}$ as the number of the shortest temporal paths between node s and z and $\sigma_{s,z}^{sh}(v)$ is the number of shortest temporal paths connecting s and z passing through *v*, the ONBRA authors focused the problem of computing an$absolute\ (\varepsilon, \eta) - approximation\ set\ of\ B(T)$ where $B(T) = \{(v, b\tilde{}(v)) : v \in V)\}$, so that $B\tilde{}(T)$ is an approximation set such that the following constrain is valid:

$$\mathbb{P}[sup_{v \in V} |b\tilde{}(v) - b(v)| \leq \varepsilon] \geq 1 - \eta$$

Briefly the approach is:given $D = \{(u, v) \in V^2 : u \neq v\}$ and $\ell \in \mathbb{N}$, $\ell \geq 2$ be a user specified parameter, then:

1. the first step is to collect ℓ pairs of nodes $(s_i, z_i)$with $i = 1, \ldots, \ell$ sampled uniformly at random from *D*.
2. for each pair (s_i, z_i) it computes the shortest Temporal Path from s to z, $P_{s,z} = \{P_{s,z} : P_{s,z} is\ the\ shortest\}$ *shortest*}
3. for each node $v \in V\ s.t. \exists P_{s,z} \in P_{s,z}\ with\ v \in int(P_{s,z})$ or in other words is computed the unbiased estimator of the temporal centrality b(***v***) $b\tilde{}'(v) = \frac{\sigma_{s,z}^{sh}(v)}{\sigma_{s,z}^{sh}}$,
4. Then finally it computes for each node $v \in V$ the unbiased estimate $b\tilde{}(v)$ of the actual betweenness centrality b(***v***) by averaging $b\tilde{}'(v)$

over the ℓ sampling steps such that $b(v) = \frac{1}{\ell}\sum_{i=1}^{\ell} \tilde{b}'(v)_i$, where $\tilde{b}'(v)$ is the estimate of b(v) obtained by analyzing the i-th sample, $i \in [1, \ell]$.

To draw a conclusion we mention the Corollary 4.2: *Given a temporal network T, the pair* $(\varepsilon', \tilde{B}(T))$ *in output from the algorithm such that , with probability* $> 1 - \eta$*, it holds that* $\tilde{B}(T)$ *is an absolute* $(\varepsilon, \eta) - approximation$ *set of B(T).*

Temporal Walk Centrality (TWC) [1]

A *Temporal Walk Weight* is $\tau_{\phi}(\omega) = \prod_{i=1}^{k} \phi(t_i + \delta.\, t_{i+1})$ with $\omega = (e_1, .., e_k)$ of a graph *G* and ϕ: $\mathbb{N} \times \mathbb{N} \to \mathbb{R}$ as a time dependent weight function. Then we need to define the incoming and outgoing *Temporal Walk* as function of $\tau_{\phi in}(\omega)$ and $\tau_{\phi out}(\omega)$, then:

- $W_{in}(v, t) = \sum_{\omega \in W_{in}} \tau_{\phi in}(\omega)$
- $W_{out}(v, t) = \sum_{\omega \in W_{out}} \tau_{\phi out}(\omega)$

To conclude in defining the *Temporal Walk Centrality:* Let $G = (V, \xi)$ be a temporal graph then $C(v) = \sum_{t_1, t_2 \in T(G), t_1 \leq t_2} (W_{in}(v, t_1) \cdot W_{out}(v, t_2) \cdot \Phi_m(t_1, t_2))$ is the temporal walk centrality of node $v \in V$, and Φ_m is a time dependent weight function for weighting the time between obtaining and distributing information at node *v*.

The overall algorithm works as follow:

1. Compute the Temporal Incoming Walks,
2. Repeat the above step for the Outgoing Walks,
3. Compute the Temporal Walk Centrality

The time complexity of the Streaming implementation is $O(|\xi| \cdot max\{\tau^{-}_{max}, \tau^{+}_{max}\})$ and for space complexity is $O(|V| \cdot max\{\tau^{-}_{max}, \tau^{+}_{max}\})$.

All the information about the Theorems, Lemma and Corollary can be found in the original paper.

Temporal Betweenness Centrality (TBC) [2]

Similar to ONBRA, what we rename as TBC, this is another approach to compute the *Temporal Betweenness Centrality;* however with the respect to the previous methods here the theoretical base relies on the problem of finding the *exact* centrality for each node. The steps are:

1. For each node of the graph, we set it as source node $s \in V$ in order to compute its predecessor graph and walk costs from s to all the other nodes, by using Bellman-Ford algorithm.
2. For all the *Predecessor Graphs*, computed in the previous step, are counted the paths; this is done by computing the strongly connected components (SCC) with the Kosaraju algorithm in order to keep only the ones with size 1; this process allows to have a remaining graph *G'* that is a *DAG*.
3. The next step is to compute the Betweenness Centrality.

From the Corollary 5.6 they states that to compute the betweenness centrality of all vertices in a temporal graph, this algorithm takes $O(n^2\, MT^2)$ where *n* are the number of nodes , *M* the numbers of edges and *T* the number of time steps.

Temporal Betweenness Centrality in Dynamic Graph (TBCDG) [4]

The version described here, and later on re-implemented and fixed in order to be used for our purposes, refers only to the *distributed* version of the algorithm in case of *sliding windows.* The approach uses an extended version of the shortest path "shortest-fastest path" dependent to a parameter α defined by the user. So if α = 1 the algorithm uses the shortest path, if α = 0 it is obtained the fastest path; the mid-range of these values results in a combination of the 2 types of temporal paths. Moreover a graph this time is defined in term of temporal windows: $G = (V, E_w)$ and the temporal path of a pair of vertices $u, v \in V$ is a sequence of edges $p(u,v) = \{(u = v_0, v_1, t_0), (v_1, v_2, t_1), \ldots, (v_n, v_{n+1} = v, t_n)\}$, *such that* $\forall i \in [1, n]$ it holds that $t_{i-1} \leq t_i$. And again as TBC and ONBRA, TBCDG computes the Betweenness centrality, as it has been defined before, now extended to the Temporal Betweenness Centrality of a vertex *v* in a windows graph G_w, that is $TBC(v) = \sum_{s,d \in V, s \neq d} \frac{\sigma_{SFP}(s,d|v)}{\sigma_{SFP}(s,d)}$, where $\sigma_{SFP}(s, d)$ are the number of Shortest-Fastest-Paths from the vertex *s* to *d in* G_w. Subsequently the considered problem is to take into account a window graph which is increasing throughout the time until a limit size, decided by the user, is reached. At each step it is collected a single *snapshot* of the temporal graph and each *snapshot* is positioned on the left of the window. In case the overall size of the window is reached, the eldest *snapshot* is erased and the one next to it will be repositioned to the left, therefore the vertices are renamed and the edges get weighted as an outcome of the transformed graph.

Next step is to process in 3 phases the computation of the Shortest-Fastest-Paths (SFPs):

1. a graph transformation is done, where the graph is turned into a static direct and weighted graph *G'(V',E',r)* with *r* the weighting function;
2. shortest paths are computed in the new static weighted graph *G'* with the application of the Dijkstra algorithm starting from a dummy source vertices,that are added from any vertex;
3. Finally, the outcomes are aggregated in order to remove duplicates.

For centrality computation is used the Brades' algorithm: it uses the concepts of dependency and pair-dependency, which are respectively: $\delta_{s\cdot}(v) = \sum_{w \in V} \delta_{sw}(v)$ and $\delta_{sw}(v) = \frac{\sigma_{sw}(v)}{\sigma_{sw}}$. At the end of Brades' algorithm all the centralities are computed, and only after the centralities are merged for each vertex in all the timestamps of the window W. The algorithm time complexity is $O(\frac{n}{|CM|}(m + n\, log\, n))$ and the space complexity as $O(\frac{n}{|CM|}(m + n))$.

Temporal Node Centrality in Complex Network (TNCCN) [5]

There are not many differences from the previous algorithm in terms of theory: it searches the shortest paths from a *direct* temporal graph with edges sorted in terms of time, and it exploits the dynamic programming approach to compute the betweenness centrality. The steps are the follows : for each node $v \in V$, it is computed $\sigma_{t,j}(s,d,v) = \sigma_{t,k}(s,v)\ \sigma_{k,j}(v,d)$
if $\Delta_{t,j}(s,d) = \Delta_{t,k}(s,d) + \Delta_{k,j}(v,d)$, where $s \neq d \in V$ and $i \leq t < k < j$. Since the Graph is acyclic we compute $\sigma_{t,j}(v,v)$ and $\Delta_{t,j}(v,v)$ between $v \in V$ and $v \in V$, where $i \leq t < j$, in no more than $O(m^2|V|^2 + |\varepsilon|)$ time. To summarize, the entire algorithm takes $O(m^3|V|^3)$ and $O(m^2|V|^2)$ in space.

Implementations

The previous paragraph lightly argues how the theory was behind these algorithms. Now, instead, we are going to show the way we have modified them in order to make them working for this experiment and be runnable in a common pattern of dataset, also current versions of programming languages are used with the respect to the original ones. Even though the most recent algorithms do not require much work, instead the oldest ones have deprecated functions and usually implied a sort of manual preprocessing for the datasets or do not have at all the algorithm. What was common, there is not an implementation of the statistics or saving methods for the outcomes computed. Therefore, we have changed how the datasets are read and some of the data structure used as a starting point of the algorithms. Then, all the algorithms require adding a method to get the TopK Node Centralities and a custom method to save the obtained results. Below are listed all the changes made in all the algorithms. Particularly, in TBC we describe in depth the new version of the algorithm and why we have decided to adopt these changes.

We have also adopted the most fresh libraries of Temporal Graph so far :

- TGLib (2022) a C++ and with Python Binding to use in both the languages, highly adaptive for very Large Temporal Graph, used for TWC, TBC and ONBRA , in this latter case just to compute the TopK Results.
- DyNetx (2020) Library created above NetworkX, used for TBCDG and TNCCN.

ONBRA

As anticipated above, the change in this case was the inclusion of the opportunity to save the resulting file in a location given by the user and, in addition, instead of saving all the centralities for all the nodes, we opted to add a method to compute the TopK Node Centralities that is used for the Statistics.

TWC

The authors were dealing with non-standard dataset; therefore, they were forced to manually manipulate the input data before starting the actual computation on them. Thus, we rewrote it through TGLib in a way to avoid unnecessary modifications of the datasets, and added, as before, the retrieval of TopK results and the method to store the final outcomes. We used Ordered Temporal Edge as the graph data structure, provided by TGLib.

TBCDG

With respect to the previous algorithm, here the Python version used at the publication was the 2.7 and it contains a lot of deprecated methods. And as before, the authors were manually preprocessing the data before the actual use for the computation of the centrality score. So, as first thing, we adopted the data structure of DyNetx to load the Temporal Graph, then for each timestamp it is created a snapshot of the temporal graph, that represents the graph at that timestamp. We have also substituted the number of loop rounds decided by the user with a loop over the full timestamp acquired automatically after reading the Temporal Graph. Then all the code was translated to fit the current standard of Python. It was also added the method to select the TopK outcomes and the method to save the results in a file specified by the user.

TNCCN

Without distancing too much from TBCDG we re-freshed the whole algorithm to the current Python version, and from the fact that in the original code lacks completely of: dataset reading; automating building of Temporal Graph and the entire data structures and methods were built from scratch; we have standardized it with DyNetx Temporal Graph representations and methods, same for the conversion from a Temporal Graph to a Dyrect Static Weighted Graph. Also the methods to pick the TopK and to save the results has been added.

TBC

This algorithm hasn't been provided with any kind of implementation, thus we needed to implement it from zero. We first want to highlight that we have decided to implement TBC with C++ in order to minimize as much as possible the computational time, due to the nature of TBC that computes the exact betweenness centrality of the nodes, which is a high time consuming task.

This is how we proceeded : we first studied the TBC paper and the new C++ library for temporal graphs, TGLib, in order to understand how to implement the algorithm and which data structures are the best to represent the data needed for the computation of the centralities.

After a long examination we have implemented the algorithm as it is proposed by the paper and tested. Unfortunately, during the test phase, we have found out that the computation of the predecessor graphs is a bottleneck, indeed we haven't been able to see the end of the computation with our small dataset used for code test purposes.

Therefore we have decided to change the Bellman-Ford algorithm with the Dijkstra algorithm, which has $O(|E|log(|V|)$ computational time in the worst case (with respect to Bellman-Ford that has $O(|E||V|)$). This change

is allowed due to the fact that the temporal graphs never present negative cycles, thus we have improved the proposed algorithm in terms of time.
We also needed to improve the algorithm in terms of memory, in fact the author version computes all the predecessors graphs (one for each node of the temporal graph) and then the node centralities are computed. This procedure makes the memory grow exponentially (in our case, during the test phase, the operating system killed the program and threw an "Out of Memory" exception).
This issue has been solved by computing for each node its predecessor graph and immediately after its centrality score, in fact to compute the centrality score of a node only its predecessor graph is requested.
This expedient allows us to save a lot of memory by keeping only one predecessor graph stored.
The data structures used in our implementation are provided by TGLib and a data structure implemented by us to represent a generic static graph, which implements a graph with an adjacency list.
To conclude, here it has been implemented the two functions to retrieve the top k results and save them in a user defined file, as it has been done in the others algorithms.

Numerical Results and Statistics

Setup
For our experiments we used a laptop with the following configurations: Intel Core i7-6700HQ CPU 4 cores @ 2.60GHz - 16 GB RAM.
Also, as software configurations we used: Ubuntu 22.04 LTS with gcc and g++ 11.02 and Cmake 3.22, while for Python we used the 10.09 version. For the full details we refer to the GitHub links at the ends of the article.

Note apart, we needed to use a different machine for the algorithm of TBCDG and TNCCN due to several heating issues. The first mentioned algorithm arose until abruptly switching off the device for hardware's protection, making impossible the completion of the task. The machine used for them has the following configuration: 13th Gen Intel(R) Core(TM) i5-13600KF 14 cores 5.08 GHz - 16 GB RAM

Datasets	$\|V(\mathcal{G})\|$	$\|E(\mathcal{G})\|$	$\|T(\mathcal{G})\|$	$\|E(G)\|$
CollegeMsg	1,899	20,296	193 days	59,835
SMS-A	44,000	unknown	338 days	5,448,000
FBWall	60,290	333,924	100 days	817,035
email-Eu-core-temporal	986	332,334	803 days	24,929
sx-mathoverflow	24,818	506,550	2,350 days	239,978
sx-askubuntu	159,316	964,437	2,613 days	596,933
sx-superuser	194,085	1,443,339	2,773 days	924,886
wiki-talk-temporal	1,140,149	7,833,140	2,320 days	3,309,592
sx-stackoverflow	2,601,977	63,497,050	2,774 days	36,233,450

Figure 2: Statistics of datasets : nodes, Temporal Edges, TimeSpan and number of Static Edges

Experiments
For this experiment we used 9 Datasets, as shown in Figure 2.

We tested each dataset on each presented algorithm, retrieved the TopK results (that in our case was chosen to be only the 10 most central nodes) and compared them with Kendall-Rank correlations.
For each dataset, the values obtained with TBC (that computes the exact centrality) were compared with the values computed by ONBRA and TWC by calculating the Mean Absolute Error (MAE) and the Maximum Deviation.

Algorithms	MAE	Maximum Deviation (from TBC)
ONBRA	21.63886	21.7388
TWC	128112.711	300269.261

Figure 1: ONBRA and TWC are matched with TBC for Mean Absolute Error and Maximum Deviation

From the outcomes of the Kendall-Rank correlation we can see more similarity between TBC and TBCDG in the CollegeMsg dataset, and higher contrast for TWC and ONBRA, slightly less for ONBRA and TBC. These differences are not much highlighted in higher datasets where ONBRA and TWC present independent and non-constant correlation close to 0.

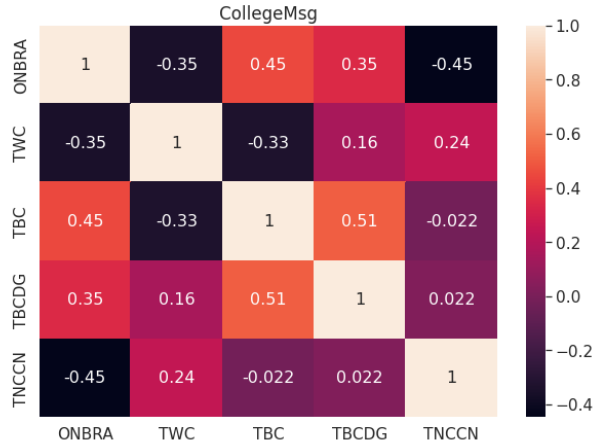

Figure 5. Kendall Rank for CollegeMsg for all the algorithms and k = 10

As confirmation for the Kendall-Rank correlation, the MAE and the Maximum Deviation computed in the CollegeMsg show that ONBRA is more accurate than TWC, with respect to TBC.

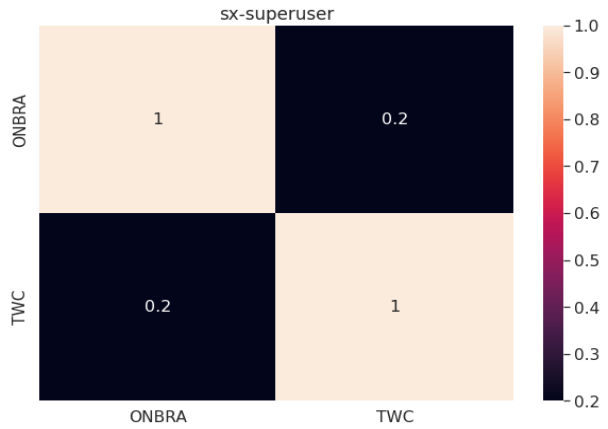

Figure 6. Kendall Rank for sx-superuser for ONBRA and TWC, k = 10

Throughout the whole experiment is notable the spaces required to run ONBRA, TBC, TNCCN that makes their execution unfeasible in a common machine without an excessive RAM installed.

To consider that:

Mem(ONBRA) < Mem(TNCCN) < Mem(TBC)

in most cases.

Datasets	$\varphi_s(ONBRA)$	$\varphi_s(TWC)$	$\varphi_s(TBC)$	$\varphi_s(TBCDG)$	$\varphi_s(TNCCN)$
CollegeMsg	12600 sec	960 sec	172800 sec	518400 sec	1200 sec
SMS-A	22680 sec	1800 sec	-	OoT	-
FBWall	28800 sec	3300 sec	-	OoT	-
email-Eu-core-temporal	38520 sec	5400 sec	-	OoT	-
sx-mathoverflow	46800 sec	23400 sec	-	OoT	-
sx-askubuntu	50420 sec	29205 sec	-	OoT	-
sx-superuser	72000 sec	32400 sec	-	OoT	-
wiki-talk-temporal	-	46800 sec	-	OoT	-
sx-stackoverflow	-	64800 sec	-	OoT	-

Figure 4: Statistics of the time required to the completion of the execution for each datasets : OoT means that the time required for the computation is out of the possibility to be done, - instead is unknown due the OoM exception.

Only TWC was able to carry to a conclusion all the text without any OoM exception. Due to the time infeasibility of TBCDG, it was not possible to demonstrate the power of a distributed version for the computation of the temporal centrality. In fact, it was the one with the worst time complexity even in the smallest dataset.

We know that there is some improvement in using Scala or Java instead of Python, which is not fairly comparable in terms of time. But from a not so recent research, it was proved that even a Scala version of Apache Spark MapReduce was not able to reach the same performance of a MapReduce with OpenMp in C++ [6]. With these statements we can conclude that TWC also in practice outperforms most of the current approaches.

To note that ONBRA needs a specific configuration to improve its performances in time. We use the parallelized version, to transform the temporal graph to a static direct graph and to sample from 1000 nodes in CollegeMsg, 2000 for SMS-A, 2500 for FBWall, 5000 for email-Eu-core-temporal , sx-mathoverflow and superuser. For the larger datasets, we tried all the possible sample sizes, from 100 to 10000 nodes, without any change in rules; anyway those lead always to OoM exceptions. Either TBC and TNCCN have bottleneck memory problem, which grows quadratically. However for TBC there is only the distributed version that could be thought of as possible implementation to improve the memory consumption.

Conclusion and Future Perspectives

We proposed a survey of some methods for the computation of Node Centrality in Temporal Graphs. We have shown the challenges and the inefficiencies of the exact and approximate solutions.

We implemented TBC and re-wrote the old approaches. However we believe that one of the future analyses to work on is the comparison of the 2 most recent approximated approaches such as ONBRA and TWC and re-implement them in a Distributed manner reducing memory complexity and see if they can be improved in time. Finally, a future research could be the exploration of Graph Neural Network to solve the Node Centrality problem.

References

[1] Lutz Oettershagen, Petra Mutzel, and Nils M. Kriege. 2022. Temporal Walk Centrality: Ranking Nodes in Evolving Networks. In Proceedings of the ACM Web Conference 2022 (WWW '22). Association for Computing Machinery, New York, NY, USA, 1640–1650. https://doi.org/10.1145/3485447.3512210

[2] Rymar, M., Molter, H., Nichterlein, A., Niedermeier, R. (2021). Towards Classifying the Polynomial-Time Solvability of Temporal Betweenness Centrality. In: Kowalik, Ł., Pilipczuk, M., Rzążewski, P. (eds) Graph-Theoretic Concepts in Computer Science. WG 2021. Lecture Notes in Computer Science(), vol 12911. Springer, Cham. https://doi.org/10.1007/978-3-030-86838-3_17

[3] Diego Santoro and Ilie Sarpe. 2022. ONBRA: Rigorous Estimation of the Temporal Betweenness Centrality in Temporal Networks. In Proceedings of the ACM Web Conference 2022 (WWW '22). Association for Computing Machinery, New York, NY, USA, 1579–1588. https://doi.org/10.1145/3485447.3512204

[4] Tsalouchidou, I., Baeza-Yates, R., Bonchi, F. *et al.* Temporal betweenness centrality in dynamic graphs. *Int J Data Sci Anal* **9**, 257–272 (2020). https://doi.org/10.1007/s41060-019-00189-x

[5] Kim, Hyoungshick and Anderson, Ross, et al. 2012 Temporal node centrality in complex networks, In Phys. Rev. E vol 85, American Physical Society. https://link.aps.org/doi/10.1103/PhysRevE.85.026107.

[6] Li, Junhao et al. 2018 Comparing Spark vs MPI/OpenMP On Word Count MapReduce, arXiv, https://arxiv.org/abs/1811.04875

[7] Oettershagen, Lutz and Mutzel, Petra et al. 2022 TGLib: An Open-Source Library for Temporal Graph Analysis , arXiv, https://arxiv.org/abs/2209.12587

Repository

The repository of the project is at the following link : bprin97/A_Survey_on_the_Different_Approaches_to_Temporal_Node_Centralities (github.com)