Graph Analytics on Proximity Data to Fight Contagion

Snehasis Banerjee[a], Vivek Chandel[a] and Avik Ghose[a]

[a] TCS Research & Innovation, Tata Consultancy Services, Kolkata, India

Abstract

As social distancing has become the new normal, it is eminent for enterprises and administrations to make decisions based on analytics on human proximity data. Graph Analytics can serve as an essential technology to understand human network behaviors and contact tracing inferences in pandemic and post-pandemic scenarios. A large amount of work has happened on epidemiology of pandemics. However, a formal application of graph analytics backed by visualization and mapping of graph algorithms has remained neglected in pandemic scenarios. This paper has addressed this important need backed by graph modeling, methods and analytics.

Keywords

graph analytics, contact tracing, COVID-19 pandemic, human proximity data

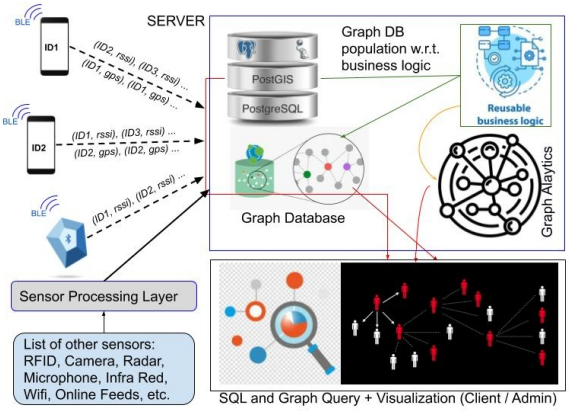

Figure 1: System Architecture for enabling Graph Analytics

1. Background

Social distancing [1] has become an essential need to manage pandemics like COVID-19. In recent times, the world has witnessed deployment of multiple apps and systems by governments [2]. Contact tracing and related methods are expected to remain prevalent in a post pandemic scenario. Adoption in enterprise is expected to be of the order of mass scale. However, the enterprise level settings will be different from a government monitored contact tracing scenario. As an example, in a factory, same device may get shared across multiple workers in rotation. Also, only person to person proximity events will not be futile, as, an infected person can be at a location for a certain time which is visited by another person after some time – this events will be missed in standard proximity detection solutions. Also, depending on the infection type (in this case COVID-19), different persons and places will have different levels of susceptibility to risks, based on the infection's properties and incubation period. Hence it is important to have: (a) a graph model to store and represent proximity events and related attributes, that is robust and scalable (b) fast techniques that can aid in contact tracing (c) methods that can help risk estimation by applying graph analytics. This paper presents a system and several methods to tackle the prior points.

As seen in Fig. 1, the system intakes proximity relation data (either person to person or person to zone) from a multi-modal sensory input. It can be GPS (Global Positioning System) and/or Bluetooth Low Energy (BLE) present in contact tracing smart devices (like phone and watches) or it may be a multitude of hard sensors like BLE and LoRa (long range low-power network communication) beacons or Radio-frequency identification (RFID) tags. Crowds (susceptible places of high proximity events) at a specific location can be detected by radar, microphone captured sound level, infrared, Wifi signal as well as soft sensors like location tagged social media feeds. Sensor feeds are stored in a relational database management systems (RDBMS, such as PostGres with PostGIS [1] having spatial indexing support) in form of proximity event tuples : { *entity, entities detected, timestamp, optional attributes*}. Based on pre-set business logic (like duration of aggregation), data from RDBMS is updated periodically in a graph database (like OrientDB[2]). Complex queries and graph analytics are run on the graph from the front-end application to get insights and visualization.

[1] https://www.postgresql.org ; https://postgis.net
[2] https://www.orientdb.org

Title of the Proceedings: Proceedings of the CIKM 2020 Workshops, October 19-20, Galway, Ireland.
Editors of the Proceedings: Stefan Conrad, Ilaria Tiddi.
snehasis.banerjee@tcs.com (S. Banerjee)
0000-0001-6497-2085 (S. Banerjee)

2. Graph Data Population

While the individual events of proximity instances are stored in a RDBMS, the actual inference and analytics happens over the graph database. As any active sensor consume power, hence keeping a sensor device always online which sends events to server is impractical. Hence, there is a sleep and wake cycle, when proximity events are detected and sent to server database at intervals. However, to perform graph analytics, the data instances need to aggregated to populate the graph with meaningful information. Therefore, if there is no gap greater than a preset time threshold T in terms of two persons being in contact, the proximity event instances becomes a continuous event. When the gap is more than T, a new entry is entered in the graph in a similar fashion. Sometimes, due to sensor errors and signal reflection, false proximity events can be recorded. However, as this is two way detection (person 1 detecting person 2 and vice versa when they are in close contact of say 2 metres) [3], if there is only one event detected, that is dropped from graph insertion – this is a configurable setting as per sensitivity needs of specific deployments. In order for wake and sleep cycles across sensors to be consistent, a standard time gap is maintained across devices, which is typically a sleep of 30 seconds and a wake of 5 seconds synchronized to 00:00:00 hours of UTC (Coordinated Universal Time). The data in relational database and graph is deleted based on some pre-set expiry timestamp to maintain a sliding window based on business logic or infection risk period (usually 30 days for COVID-19 scenario). Geo-spatial queries may be directly run on spatially indexed graph database or be optionally run on the spatial layer of the relational database (to use spatial geometric algorithms) by entangling with graph queries, however, in enterprise scenarios (small geo-location coverage), such needs are found to be rare.

2.1. Entities as Graph Nodes

A necessary part of graph modeling is identification of entities and its representation for fast graph queries and traversal. Here, the nodes are of 4 types:

(1) Person: humans having features (attributes) like age, risk factors that remain static over a period of time. Dynamic attributes are derived or calculated over an interval – risk estimation, proximity activity and health estimate. A necessary attribute of a person in a contagion scenario is the person's current health status - infected, healthy or susceptible.

(2) Person tagged sensor: sensors like wearable smart devices and smartphones are tagged to a person – that

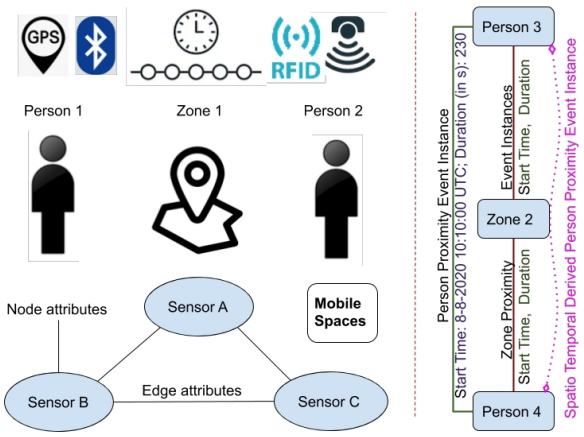

Figure 2: Modeling of nodes and edges in a proximity graph

is how a person proximity event is generated. A person can have multiple tagged sensors (like two smart phones). Same sensor device (with a unique device ID) can be used by various persons in disjoint time frames.

(3) Static zone tagged sensor: fixed sensors like beacons in elevators, office areas, cabins, canteens, gym, shops, etc. helps in proximity event detection in two ways: (a) Any detected proximity event confirms positively if two or more persons were already reported by person tagged sensors. (b) Usually infectious diseases stay active at a space for some period of time. Therefore, a proximity event with a person at time T1 and another person with time T2 can trigger an indirect proximity link between the two persons, if $|T1 - T2|$ are within the infectious period relevant to that space.

(4) Dynamic zone tagged sensor: sensors tagged with transport can keep track of proximity events that occur when a person stays in that zone. It is important to keep track of the time span a person is in this dynamic zone, as the entry and exit times will reveal the spatial zones of entry and disembarkment. This needs to be matched with other persons having overlapping proximity events but different entry or exit points. Contagion can be tracked in the related entry and exit spaces (like bus stops, airports, parking lots, stations) even if those spaces have bad sensor coverage.

Locations can be sensed from GPS or cellular tower estimates in outdoor settings. In indoors, various localization techniques leveraging radar, RFID, Wifi, BLE can be used. There are other possible node types specifically mobile objects (like parcels, posts, shopping items, currency notes) that can play a part in the infection spread. However, as proper sanitization practices can mitigate these occasional object based contagion risks, it is kept out of scope of the paper.

2.2. Proximity Events as Graph Edges

As presented in Fig. 2, a new proximity event is entered into the graph as an edge with start time (preferably UTC timestamp) and time duration (say seconds) as necessary attributes. In the graph, relations are maintained between persons and not devices (bearing sensors) as same device may be used by different persons over a period of time. So person and zone resolution at the time of graph population is of high importance – proximity event edge keeps an attribute source (like device sensor) to handle any future backtrack queries. Indirect proximity events are inferred using transitive relation of person and zones with timestamp constraints. By nature humans are mobile, hence it makes sense to keep a private record of all other nodes (places and other persons) visited within a sliding interval of time period as edges. Because, two nodes in this graph can have multiple edges (proximity instances) and proximity events are by definition both way, the resulting graph is an *undirected Multigraph.* Due to social distancing measures, this graph will be sparse and have local community clusters. Hence, adjacency list is the best way to represent the relations between |V| vertices (nodes) and |E| edges, resulting in 2*|E| space complexity. A hash index on the nodes will aid in O(1) lookup and fast adjacent node traversals. In a sparse graph with community clusters, graph traversal using breadth first search is the optimal approach, which takes time around $(V_c + E_c) * d_l/D_c$, for a depth up to level d_l. V_c and E_c are respective vertices and edges at a community cluster in the graph of cluster diameter D_c (largest edge distance or number of nodes in path between two nodes in that cluster or sub-graph).

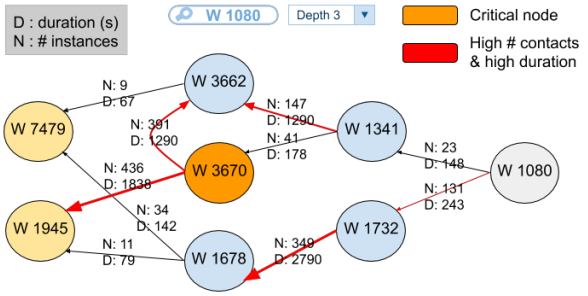

Figure 3: Visualization of proximity relations (with number of interactions and duration) given a person and depth level

3. Graph Analytics

Graph analytics deals with pairwise relationship between two entities at a time; as well as structural characteristics of the graph as a whole. Graph Analytics also includes visualization aspects, insights gained from complex query execution and graph algorithms.

3.1. Graph Queries and Visualization

In contact tracing scenario, one of the primary requirement is to find the chain of persons an infected person has come across. This can be done by graph traversal and the results can be showed in front-end visualization as shown in Fig. 3. For privacy reasons, the exact identity of a person is not disclosed and an encrypted alias is used. Other related graph queries are time overlap between visits to specific zones, the total duration of contact, the number of instances of contact, number of infected visiting a specific zone and the time spent there. Other queries can detect crowds at locations based on large number of temporally colliding proximity events – those zones need to be marked as 'at risk' zones. Crowd detection will be based on spatial queries on spatio-relational database (say PostGIS) if dealing with latitude and longitude values (GPS).

3.2. Centrality Measures

In managing pandemics and enforcing social distancing, it becomes crucial to find the specific zones that are hotspots and which persons have highest capability to spread infection. Usage of graph centrality measures helps in identification of critical nodes.

Degree centrality, measured by the number of unique edge links of a node to other nodes can be thought of the relative risk of getting infected if the local community sees an infection outbreak.

Closeness centrality of a node is: C(i) = $\sum d_{ij}$, where d_{ij} is the geodesic distance from node i to node j (number of links in the shortest path from node i to node j). Closeness reveals how long it takes for infection to flow from one node to others in the graph.

Betweenness centrality signifies that a node is crucial or forms a bridge for spreading of infection. Examples are person nodes who visit many other nodes like doctor, courier and security staff. Examples of bridge space nodes are market and transport zones. Betweenness centrality is measured as, b(i) = $\sum g_{jik}/g_{jk}$, where g_{jk} is number of shortest paths from node j to node k (j and k ≠ i) and g_{jik} is number of shortest paths from node j to node k passing through node i.

There are other graph centrality measures [4] like Eigenvector Centrality. However, it requires calculations over adjacency matrix where as the graph model shown is based on adjacency list – hence not optimal.

3.3. Graph Structure Measures

Proximity events between 2 persons represent a Dyad in graph, where as proximity event between 2 persons and a mutually visited zone forms a Triad. This are the simplest forms of graph structures.

Eccentricity is the maximum distance from a given node to all other nodes in a graph. The diameter of a graph is the maximum eccentricity value. Eccentricity calculation aids in understanding contagion's spread from one node to the others. If infection propagates from a node A towards some specific nodes in the graph in the smallest number of steps, it means that the node A has better propagation efficiency.

Density serves as an important graph structure measure, as more dense a graph is (more connected nodes), the more the risk of infection spreading. Density is calculated as: $D(G) = 2 * |E|/(|V| * (|V| - 1))$.

A clique is a graph (or subgraph) in which every node is connected to every other node. Finding maximum cliques in a graph provides the largest set of common links – hence, strong communities can be discovered with the following risk – if any one is infected, it will result in a high probability of others also getting infected. Another structure, K-cores, which are not necessarily cohesive groups, but indicate areas of a graph which contain clique-like structures.

Clustering coefficient measures the drift of nodes to form dense subgraphs. The clustering coefficient C of a node i shows how its neighbors are connected with each other. The local (node) clustering coefficient is calculated as follows: $C_i = 2*T_i / (K_i * (k_i - 1))$, where T_i is the number of distance triangles with node i and $K_i * (k_i - 1)$ is the maximum number of possible connections in neighbors of i. A large C implies that the graph is well connected locally to form a cluster. Average graph clustering coefficient is: $C = 1/n * \sum C_i$.

3.4. Community Detection

In contact tracing and related analytics, it is important to discover communities from the proximity events so that at risk communities can be identified and infected communities can be kept in isolation. As seen from Fig. 4, number of communities increase with the number of proximity event instances. Also the density of the communities increases reflecting closely connected contacts. The task of community detection (or graph clustering) is to discover subsets of nodes (clusters) of connected communities in which nodes have many internal edges and few external edges. Detecting communities in a graph is NP-complete. Community detection methods can be divided into three categories:

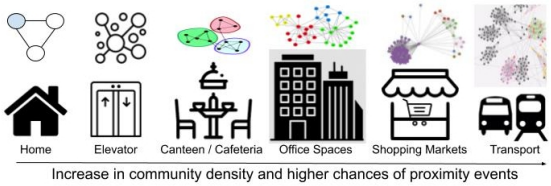

Figure 4: Importance of Community Clusters in Contagion

divisive, agglomerative, and optimization algorithms. Some popular algorithms [5] are K-cliques, hierarchical clustering, spectral bisection, graph partioning, modularity optimization and link communities. One of the methods suitable for large graphs is Louvain Method [6] that uses graph modularity and has time complexity as O(|V|log|V|). Modularity is a measure of the structure of the graph and is defined as, $Q = (|E_{in}| - |E_{in-R}|) / E$, where $|E_{in}|$ is the number of links connecting nodes that belong to the same community and $|E_{inaR}|$ is the estimated $|E_{in}|$, if links were random. The Louvain Method works as follows: (1) By optimizing modularity locally on all the nodes, small communities of nodes are found. (2) Small communities are then grouped into one node, and the first step is repeated.

3.5. Contagion and Risk Propagation

Some work has been done on contagion and risk estimation [7] [8]. However, they are dependent mostly on external factors to estimate the risk. Here, a method is presented that uses a combination of graph modeling, graph structure measures and infection properties to estimate risk for a person or community. Contagion is dependent on physical proximity, hence network diffusion techniques can be helpful here. A related problem is Link Prediction: the task here is to predict whether there will be a link between two nodes, provided that link does not pre-exist. Node similarities can be found using measures [9] for analyzing the proximity of nodes in a graph such as a degree distribution, common neighbors, preferential attachment, Jaccard coefficient and Leicht-Holme-Newman Index.

Highly influential nodes (hubs) in the graph (as highlighted in Section 3.2) enable a swift spread of an infection through the graph. On the other hand, poorly connected or periphery nodes would relax the spreading of infection and permit only a tiny portion of the graph to get exposed to the diffusion. Graphs with lots of localized clusters (in case of highly fragmented low-density graph) may limit the spread of an infection and will face hurdles in establishing any momentum, whereas dense graphs with few gaps (or few boundary

spanners) would assist the dissemination of contagion.

Earlier discussions in the paper were around finding risks in the graph structure, nodes and communities. Certain techniques like belief propagation and label propagation are popular in general graph analytics, however in this scenario, what is needed is a knowledge wrapper on breadth first algorithm which can match the scale of incoming proximity event streams. Finally the paper presents an approach to propagate risk across proximity graph (Algorithm 1). Due to storage of last propagation update attribute at each connected node, if there is a sudden stop due to server load or time constraints, the risk propagation can start from last checkpoint. This avoids updating the same node risk attribute more than once. The risk estimate propagation deals with three damping factors: x – takes care of contagion speed and is determined by disease stage of infected person (incubation, prodromal, illness, decline and convalescence), y – to handle influence of selected centrality measure, z – to handle relative community density's impact on risk.

In conclusion, the paper described various aspects of Graph Analytics for proximity graph based contact tracing in pandemic scenarios.

References

[1] M. J. M. Chowdhury et. al., Covid-19 contact tracing: Challenges and future directions (2020).

[2] N. e. a. Ahmed, A survey of covid-19 contact tracing apps, arXiv preprint arXiv:2006.10306 (2020).

[3] V. Chandel, S. Banerjee, A. Ghose, Proxitrak: A robust solution to enforce real-time social distancing & contact tracing in enterprise scenario, CPD Workshop, Ubicomp 2020 (2020).

[4] K. Das, S. Samanta, M. Pal, Study on centrality measures in social networks: a survey, Social network analysis and mining 8 (2018) 13.

[5] Y. Zhao, A survey on theoretical advances of community detection in networks, Computational Statistics 9 (2017) e1403.

[6] S. Ghosh et. al., Distributed louvain algorithm for graph community detection, in: IPDPS 2018, IEEE, 2018, pp. 885–895.

[7] N. Fenton et. al., A privacy-preserving bayesian network model for personalised covid19 risk assessment and contact tracing, medRxiv (2020).

[8] T. Bhattasali, Pandemic analytics to assess risk of covid-19 outbreak (2020).

[9] V. Martínez et. al., A survey of link prediction in complex networks, ACM computing surveys (CSUR) 49 (2016) 1–33.

Result: Risk Propagation to a specific depth under specified time constraint.

Parameters:

V ⟵ set of vertices or nodes in the graph G;

E ⟵ set of edges or links in G;

A ⟵ external inputs like node health (A_h) and attributes like degree; specified with subscript;

A_l ⟵ attribute to store last risk propagation node label;

software ⟵ (.) link to software modules and libraries;

M ⟵ diameter of graph, i.e. maximum eccentricity value;

x, y, z ⟵ damping factors to balance risk propagation bias;

Begin:

D ⟵ 0 // current graph traversal depth level;

v ⟵ V(i) ⟵ node i of set V is infected; hence A_h(i) = infect;

v(A_r) ⟵ 1 * x * w // infected person with weighted risk w;

Q ⟵ queue is created and intitialized with V(i) as root node;

R ⟵ currrent root node at depth level D;

while *Q ≠ empty* ***And*** *time constraint = satisfied* **do**

  v(A_l) ⟵ V(i) label //helps recover from sudden halts ;

  v ⟵ visited //if node is not visited earlier;

  v ⟵ Q.dequeue( ) /* Removing vertex v from queue whose neighbour will be visited next */;

  D ⟵ D + 1 //increment depth ;

  v(A_r) ⟵ R(A_r) * (1 - D / M) //risk propagation;

  if *v has relative Low Centrality Measure* **then**

    v(A_r) ⟵ y * v(A_r) //y is a damping factor;

  end

  if *v is part of a relative Large Community* **then**

    v(A_r) ⟵ z * v(A_r) //z is a damping factor;

  end

  foreach *neighbors W of v ∈ G* **do**

    if *w ≠ visited* ***And*** *w(A_l) ≠ v(A_l) //checks infected node label matches for previous halts;* **then**

      Q.enqueue(w) //Stores w in Q;

    end

  end

end

Algorithm 1: Contagion Risk Propagation