Journal of Computer Science

Original Research Paper

COV19-Dijkstra: A COVID-19 Propagation Model Based on Dijkstra's Algorithm

[1]Arnaud Watusadisi Mavakala, [2]Wilfried Yves Hamilton Adoni, [3,4]Najib Ben Aoun, [2]Tarik Nahhal, [3,5]Moez Krichen, [3]Mohammed Y. Alzahrani and [6]Franck Mutombo Kalala

[1]African Institute for Mathematical Sciences, Senegal
[2]LIMSAD Laboratory, Hassan II University of Casablanca, Morocco
[3]College of Computer Science and Information Technology, Al Baha University, Saudi Arabia
[4]REGIM-Lab: Research Groups in Intelligent Machines, University of Sfax, Tunisia
[5]ReDCAD Laboratory, University of Sfax, Tunisia
[6]University of Lubumbashi, Lubumbashi, Democratic Republic of Congo

Article history
Received: 08-06-2022
Revised: 21-11-2022
Accepted: 07-12-2022

Corresponding Author:
Najib Ben Aoun
College of Computer Science and Information Technology, Al Baha University, Saudi Arabia
Email: najib.benaoun@ieee.org

Abstract: The presence of the coronavirus, known as COVID-19, has prompted several researchers to study the mode of spread and the different defense mechanisms of the virus. As a reminder, obtaining a vaccine, for which much research is being conducted around the world, is a long and expensive process and it is unlikely that the pandemic can be treated in time. In this article, we present a new way to assess and limit the spread of the virus while trying to answer the following important questions: How to use the shortest path algorithm in a graph to analyze and better understand the spread of the virus? How to use the predictive power of the graph using the shortest path algorithm to find the relationships of a person who might be most at risk? The designed algorithm simulates how the virus spreads and infects people through the graph. Since the size of the collected COVID-19 data can reach a large volume over time and speaking of the graph concept, the NOSQL database including Neo4j which is a graph oriented NOSQL database is used for data collection, storage and processing. To enable the design and optimization of virus defense systems, this study proposes a feasible approach to quantify and predict the danger of a virus infection within a community.

Keywords: Coronavirus, COVID-19, Propagation, Graph Search Algorithm, Dijkstra, All-Shortests, Nosql Database, Neo4j

Introduction

In December 2019, the first victim of Severe Acute Respiratory Syndrome Coronavirus 2 (SARS-CoV-2), the virus responsible for the 2019 coronavirus pandemic (COVID-19), was diagnosed in Wuhan, China (Andersen *et al*., 2020). In the weeks leading up to the diagnosis, the disease spread widely in mainland China and other countries around the world, causing global panic and resulting in the first major coronavirus pandemic in a few months. However, long before SARS-CoV-2, there was SARS-CoV-1 (Giannis *et al*., 2020; Ru *et al*., 2020), responsible for the SARS outbreak in 2002-2004 and MERS-CoV from the Middle East and South Korea, responsible for the MERS outbreak in 2012 (Cai *et al*., 2013; Giannis *et al*., 2020; Liang, 2020). Unlike these two viruses, a problematic feature of SARS-CoV-2 is the existence of asymptomatic infections (Giannis *et al*., 2020). These asymptomatic infections are unaware of their ability to spread and thus lead to an increase in the number of infected persons. This has contributed to the rapid and global spread and the resulting drastic restrictions on daily life around the world to block the spread of the virus, such as the use of certain vaccines, masks, social distancing (Lewnard and Lo, 2020), city closures, school closures, traffic stoppages, community management and health education knowledge that has been adopted worldwide (de Bruin *et al*., 2020; Lin *et al*., 2020).

Because of the evolving epidemic (Sebastiani *et al*., 2020) and in an effort to provide a means of maximum contact tracing, we wish to address the following important issues for COVID-19:

(a) How to use the shortest path algorithm in a graph to analyze and better understand the spread of the virus?
(b) How do we use the predictive power of the graph from the use of a shorter path algorithm to be able to find the relationships of a person that may be the most risky?

The objective pursued in this study being to develop a suitable conceptual model that allows modeling the transmission process while considering the specific characteristics of COVID-19 in particular, which is different from some models that have been used to describe the spread of biological viruses or virus propagation in a network such as: Susceptible Infected Susceptible (SIS) model (Cai *et al*., 2013; Liang, 2020; Song and Hei, 2020), Susceptible-Infected-Remised (SIR) model (Cai *et al*., 2013; Chen *et al*., 2020; Sameni, 2020), the Susceptible Exposed Infectious Recovered (SEIR) model (Cai *et al*., 2013; Faranda and Alberti, 2020; Rădulescu *et al*., 2020) and the Susceptible Infectious Recovered Dead (SIDR) model (Cai *et al*., 2013). Thus, this modeling of the COVID-19 which has become a pandemic, will therefore lead to large-scale scenarios, involving a great computational complexity, which led us to use the Dijkstra algorithm, to reduce this great computational complexity.

In this study, we use Dijkstra's algorithm which is an algorithm that consists of computing the shortest paths in order to prevent and combat COVID-19 transmission for a large-scale graph. Indeed, researchers have been confronted with finding the shortest paths with the upper constraint on large graphs (Yang *et al*., 2019) and using a shortest path based method for gene analysis and prediction (Zhu *et al*., 2016). We propose both a new way to simulate COVID-19 propagation on large graphs and a COVID-19 propagation model based on the Dijkstra algorithm for analysis and prediction. Parameters such as node reinfection probability, propagation speed, propagation probability are introduced into our model to simulate Coronavirus activity with greater accuracy.

We use the graph-oriented database that is Neo4j with artificial data because of the need for hospitals and other medical services to keep their patient's data private but also, to involve all the factors affecting the spread of the virus. Thus, to get closer to real life, we use randomly generated data with the Neo4j benchmark randomized graph which thus mimics the structural characteristics of real-world networks based on communities already known a priori (Ghasemian *et al*., 2019). The artificially generated data took into account real life, as no element was discarded, such as people who are isolated or have no social life, people who have many relationships or only one relationship and the types of relationships that connect them. We consider a person to be infected in our artificially generated data, which represents an infected node in our large-scale graph and ensure that this node has at least one relationship but also a very high probability of propagation with a node at risk of infection.

Network and graph-based modeling has been used for may applications Event detection (Mejdoub *et al*., 2015; Aoun *et al*., 2011a-b; Zorzenon *et al*., 2021; Aoun *et al*., 2014; Al Hakim *et al*., 2022; Silveira *et al*., 2022). Network-based epidemic modeling has become increasingly popular, as it allows us to describe not only the impact of individual behavior on the spread of infection, but also to determine the best strategies for mitigating the impact of infectious diseases. When considering large-scale scenarios, there is a great deal of computational complexity involved. Thus, (Zorzenon *et al*., 2021) proposed a new notion of graph called contagion graph where, they proposed a graph-based method, derived from Dijkstra's algorithm, which allowed them to reduce the computational complexity of a simulation. This contagion graph was used as an approximation scheme describing the average behavior of an epidemic on a network and requiring low computing power. Therefore, it is important to point out that the Dijkstra algorithm mentioned earlier, is an algorithm whose concept is to determine the minimum service on certain routes and to produce sets of solutions (Al Hakim *et al*., 2022). The Dijkstra algorithm can determine the shortest path in the search for maritime traffic, for example, as in the research (Silveira *et al*., 2022) where the researchers tried to determine the shortest Dijkstra path to characterize in a systematic way the maritime traffic of the continental coast of Portugal but also, between two other borders identified as potentially interesting by observing the traffic density maps. The modeling of this approach consisted of constructing a graph in which the nodes are cells of a grid covering the geographical area under study and the weights of the directional edges that are inversely related to the movements of ships between the cells.

Currently, the modeling of a complex system such as the spread of the COVID-19 infection is among the topical issues despite the fact that several other modeling cases to help prevent and control the transmission of COVID-19 have been proposed since the outbreak of the pandemic (Block *et al*., 2020; Hope *et al*., 2020; Ordun *et al*., 2020; Yilmaz *et al*., 2020) and, the majority of his proposed models on the spread of COVID-19 infection have emphasized the prominent role of direct human-to-human transmission in the COVID-19 epidemic (Chan *et al*., 2020; Sahin *et al*., 2020; Yang and Wang, 2020). But, despite the existence of a large number of proposed models, the majority do not take into consideration the epidemiological characteristics of COVID-19 such as social distance, contact time with the infected person. Thus in (Alguliyev *et al*., 2021), they proposed a graph-based modeling of the spread of COVID-19 infection where they reviewed the studies related to the modeling of COVID-19 pandemic and analyzed the factors affecting the spread of the disease and its main characteristics while taking into account the social distance, duration of contact with an infected person and its local demographic characteristics. They developed a graphic model of the process from the first confirmed case of infection to the transmission of the virus from human

to human and visualized it considering the epidemiological characteristics of COVID-19. Unfortunately, they could not take into account all the factors that influence the increase in the number of cases of infection such as social, economic and demographic factors, population density, etc. In this study, we did not exclude any of these factors that were cited as influencing the increase in the number of cases of infection for our modeling.

The size of COVID-19 data collected from all over the world keeps growing over time. Therefore, Khashan *et al.* (2020) had proposed a storage framework capable of handling SQL and NOSQL databases which they had renamed COVID-QF for COVID-19 datasets to address and manage the problems related to the spread of the virus worldwide while reducing the processing time. The graph-oriented database that is Neo4j, which is a NOSQL database, has enabled several researchers to model and study tens of billions of nodes, properties and relationships in a single graph to address and manage real-world problems.

Problem Statement

In this section, we introduce and define the graph elements necessary for modeling the propagation of the virus. The important elements that will be used throughout this study.

Let $G = (V, E)$ a directed graph, where V is the finite set of vertices and $E = \{(v_i, v_j) \backslash v_i, v_j \in V\}$ is the set of edges connecting the pairs of vertices of V. A weight can be assigned to an edge, in this case the graph $G = (V, E, W)$ becomes weighted where W: $E \rightarrow \mathbb{R}$ is a weighting function associated with the edges of E. Representing a community of people in the form of the G graph, we have:

- For all $v_i \in V$, v_i represents the person i^{th} in the so-called community graph
- Each of the edges $(v_i, v_j) \in E$ represents the relationship between person i and person j in the community graph
- A sub-graph of G is defined as a sub-community of people in the community graph G. We have three sub-graphs: Sub-graph of infected people (I), subgraph of uninfected people (U) and sub-graph of people at risk of infection (S)
- The outer half degree of the v_i vertex of G noted by $d_G^+(v_i)$ which is the number of arcs from v_i. So, it is the number of people the person i knows in the community graph
- The inner half degree of the vertex v_i noted by $d_G^-(v_i)$ which is the number of arcs ending in v_i. So, it is the number of people who know the person i in the community graph
- The degree of v_i in G noted by $d_G(v_i) = d_G^+ + (v_i) + d_G^-(v_i)$ which is the number of arcs admitting v_i as end or neighbor. So, it is the number of relationships of the person i in the community graph

The propagation of the COVID-19 virus within a community is spread from person to person by way of aerosols, droplets that are expelled when an infected person talks, sneezes or coughs and can end up on objects that are touched and then by touching their face. We take $G = (V, E, W)$ a non-oriented graph (Fig. 1) because we do not know the different directions of the arcs on the existing relationships between people, for example who is a friend of whom in the community, such that for any $v_i \in I$, $v_j \in U$ and $v_s \in S$, the person i can transmit the disease to the close person j and the close person s under certain probabilities of propagation the virus ($P_{Propagation}$) within the community, as shown in Fig. 2.

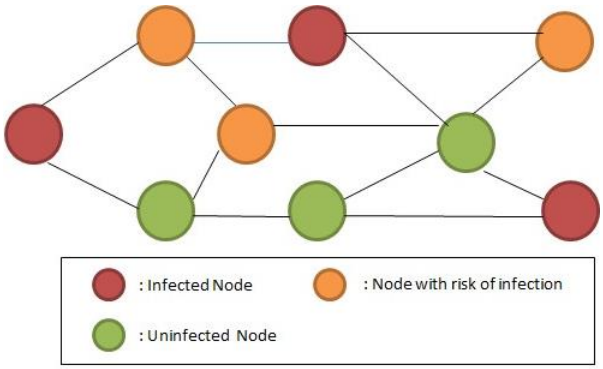

Fig. 1: Community network

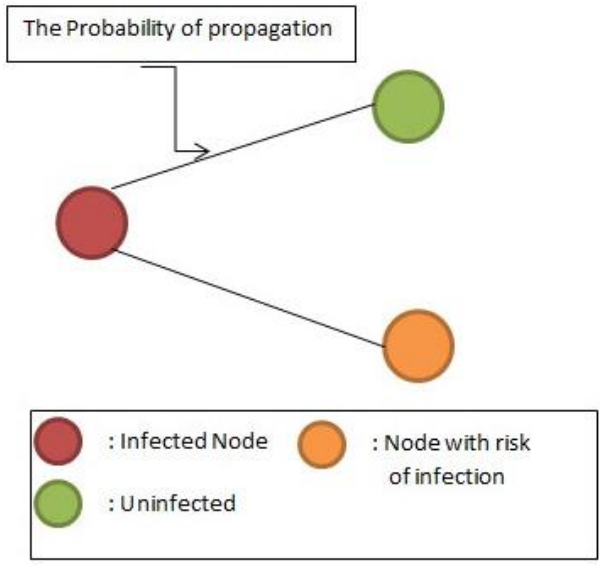

Fig. 2: Propagation of the virus

This propagation of the virus through a graph is done with a certain speed. Thus, it is said that the speed of virus propagation in the community graph is defined as follows:

$$Speed_{propagation} = \frac{N_{InfectedNodes}}{t}$$

where, $N_{InfectedNodes} = |I|$ is the number of infected nodes in the graph and t is the time taken to infect the entire graph. Following the propagation of the virus, we say that the probability $P_{InfectedNodes}$ gives us the probability of infected nodes in the graph whose formula is:

$$P_{InfectedNodes} = \frac{N_{InfectedNodes}}{N}$$

the probability $P_{UninfectedNodes}$ for uninfected nodes is:

$$P_{UninfectedNodes} = \frac{N_{UninfectedNodes}}{N}$$

and for the probability $P_{RiskNodes}$ of nodes with risk of infections is:

$$P_{RiskNodes} = \frac{N_{RiskNodes}}{N}$$

where, $N = |V|$ is the total number of nodes, $N_{UninfectedNodes} = |U|$ is the number of uninfected nodes and $N_{RiskNodes} = |S|$ is the number of nodes with risk of infections. An infected node thus transmits the virus to its uninfected immediate neighbor and once the neighboring node becomes infected, it too will continue to transmit the virus to its uninfected immediate neighbors and so on in the graph.

Background

In this section, we explain and give the principle of path finding with the single-source Dijkstra algorithm. Then, we discuss the graph-oriented database that is Neo4j and its Extended Properties Graph Model (EPGM) (Junghanns *et al*., 2016) which is an extension of a model belonging to the graph family, namely the Property Graph Model (PGM) (Rodriguez and Neubauer, 2010).

Path Computing with Dikjstra's Algorithm

The algorithm was developed by computer scientist Edsger W. Dijkstra in 1956. It allows to find from a starting node and a destination node which are both known, a shortest path between these two nodes, or to find from only one known starting node, all the shortest paths from this node to all the other nodes of the graph, we speak then about "unique source". Dijkstra's single source algorithm therefore consists of defining the starting node and finding the shortest weighted path to all other nodes in the network.

The principle of path searching with the single-source Dijkstra algorithm (Fig. 3) can be described as follows.

Step 1: Initialization

Set of vertices $V' \leftarrow \theta$ and of edges $E' \leftarrow \theta$; begin by setting $dist\ (v_i) \leftarrow \infty$ for each vertex $v_i \in V$.

The initial vertex $dist(v_{initial}) \leftarrow 0$; the i^{th} vertex is the initial $v_i \leftarrow v_{initial}$.

Step 2: Adding Vertices in V

$v_{current} \leftarrow$ Choose the minimum distance between all $\leftarrow \leftarrow V' \cup \{$vertices and $v_{current}\}$:

Step 3: Vertex Expanding

For each edges $(v_{current}, v_{i+1}) \in E$; if $dist\ (v_{current}) + w\ (v_{current}, v_{i+1}) < dist(v_{i+1})$, then update $dist(v_{i+1}) \leftarrow dist(v_{current}) + w\ (v_{current}, v_{i+1})$ and add the edge $E' \leftarrow E' \cup \{(v_{current}, v_{i+1})\}$

Step 4: Stopping Criteria

If $V' = V$ then the path with all other vertices has been found; otherwise, if $V' \neq V$ then failure; otherwise go to step 2.

The algorithm works with two main sets: V which contains all the vertices of the graph and V which contains the promising vertices of the subgraph covering the path. The output of this algorithm consists in obtaining a sub-graph that covers all the shortest paths, in which the different vertices v_i are classified in ascending order of their minimum distance to the starting vertex. During each iteration, a vertex of minimum distance is chosen outside the sub-graph and added to the subgraph and the distances of the vertices adjacent to the one added are updated by doing dist $(v_i+1) \leftarrow dist\ (v_{current}) + w\ (v_{current}, v_i + 1)$.

The single-source Dijkstra algorithm guarantees to find the optimal path if for each edge $(v_{current}, v_{i+1}) \in E$, the triangular inequality is verified: $dist\ (v_{current}) + w\ (v_{current}, v_{i+1}) < dist\ (v_{i+1})$. The complexity of this algorithm is such that the initialization itself takes $O(|V|)$ of time and the extraction of the minimal vertex also takes $O(|V|)$ of time. For the search of the successor vertices, it takes $O(d^+(v_i))$ of time. Therefore, the overall complexity of the Dijkstra algorithm is $O(|V|^2)$.

Neo4j

According to Neo4j documentation, Neo4j is an open-source database written in Java and Scala, published under a dual model of free software and commercial license (Kan *et al*., 2017; Needham and Hodler, 2019; Jabri, 2020). It consists in implementing generic property graph models and providing complete database features, including Atomicity, Consistency, Isolation, Durability (ACID) transaction compliance, cluster support and execution failover. As indicated in Fig. 4 on the Neo4j architecture, Neo4j stores graphic data in a number of

different storage files. The main components of this architecture are its REST-based API, cache, transaction log and log files (Kan *et al*., 2017).

Each component of the architecture has its own importance which shows how the Neo4j database works. We say that: The Traverser API therefore allows the user to browse the graph with the help of reminders. Indeed, the user can define an approach to search for a graph or sub-graph using specific rules and algorithms. Caching affects read and write performance. Transaction logs, on the other hand, maintain all events and operations that occur on the Neo4j database and log files are files storing information about nodes, relationships and properties.

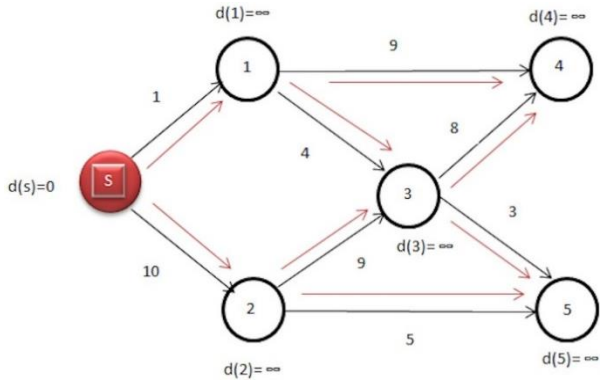

Fig. 3: Example on the Dijkstra's single-source algorithm

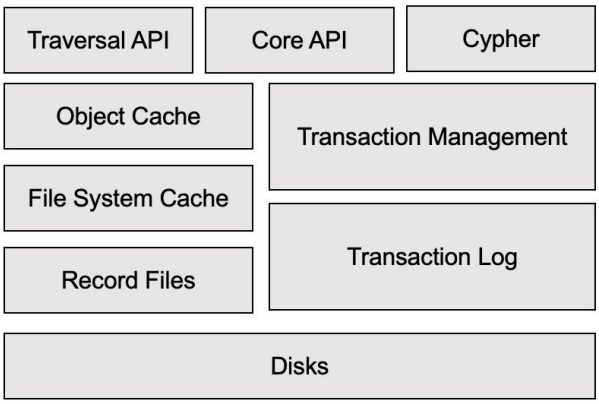

Fig. 4: Architecture of Neo4j

Scalability and Redundancy

Neo4j enables high query performance on large volumes of complex data. It can be deployed in two modes. Standalone Server: This means that one instance server is used and Highly Available cluster (HA): This means that several instance servers are used. He HA cluster is a master-slave architecture Fig. 5 which consists of a single master server and a number of slave instance servers. Each slave instance is synchronized with the master server via the cluster management component to maintain data consistency. A load balancer deployed upstream of the cluster directly manages the read/write operations of the client applications so that all read requests are distributed to the slave instances and all write requests are first synchronized with the master server and then the results are sent to the client.

To complete a write load and ensure consistency, the *HA* cluster requires a quorum to accept write operations. This means that at least $\frac{S}{2}+1$ hosts of cluster instances must be online (where, *S* is the number of servers in the cluster). If this quorum is not reached, the cluster will degrade to read-only mode. In addition, Neo4j provides auto-clustering, in case the current master server becomes unavailable, the cluster management component ensures that a new master is automatically elected. To ensure fault tolerance, the complete graph data is replicated on each instance server of the cluster and each server can host up to 34 billion nodes, 34 billion relationships and 68 billion properties.

The EPGM Model

The Neo4j database uses the Extended Properties Graph Model (EPGM) model (Junghanns *et al*., 2016) which is an extension of a model belonging to the family of graphs, namely the Property Graph Model (PGM) which was proposed in (Rodriguez and Neubauer, 2010) and which is a directed, labeled and assigned multigraph. There are data model requirements that the Property Graph Model (PGM) cannot satisfy, such as the lack of support for graph collections and associated operators. The Extended Property Graph Model (EPGM) has been proposed to meet these requirements. In this proposed model, a database consists of several property graphs in the form of key-value pairs. These graphs are generally called logical graphs and, have a type label and may have different properties.

Formally, we define the graph of the EPGM model by $G = (\mathbb{V}, \mathbb{E}, \mathbb{G}_L, \mathbb{L}, \mathbb{K}, \mathbb{A})$ where $\mathbb{V}$ a set of vertices, $\mathbb{E}$ a set of edges and $\mathbb{G}_L$ a set of logical graphs. A logical graph $\mathbb{G}_L = \langle V, E \rangle$ is an ordered pair of a subset of vertices $V \subseteq \mathbb{V}$ and a subset of edges $E \subseteq \mathbb{E}$. For labels associated with $\mathbb{V}$ and $\mathbb{E}$, $\mathbb{L}$ is used as the label alphabet T such that $(\mathbb{V} \cup \mathbb{E} \cup \mathbb{G}_L) \rightarrow \mathbb{L}$. In the same way, the properties which are of the form key-value pairs, are defined by the set K of keys and the set A of values such as $(\mathbb{V} \cup \mathbb{E} \cup \mathbb{G}_L) \times \mathbb{K} \rightarrow \mathbb{A}$, as we can Fig. 6.

COV-19-Dijkstra

In this section, we present the spread of the virus in the form of a graph and then propose a spread of COVID-19 based on the Dijkstra algorithm. We represent the entire population as an undirected G graph with nodes representing persons and we represent the infected population (I), the newly infected population, the uninfected population (U) and persons at risk of infection (S) as subgraphs of G.

We assume that a coronary patient positive or patient zero in a community represents an initial v_0 node and comes into contact with other people or other nodes on the graph that are either already infected or already healed or uninfected. Because it is an infectious disease, it spreads throughout the community and the choice of patient zero in that community is random. We propose three functions, the first two of which are constructed to simulate the spread of the virus in a graph and the third one uses instead the Dijkstra algorithm according to this mode of spread.

In order to model our different functions, we have used some parameters that are necessary to explain from the beginning such as:

- Infected which represents the list of infected nodes, therefore Infected ∈ I
- SNodes as a list of nodes at risk of infection, therefore SNodes ∈ U and S
- RNodes as a list of nodes that got the virus and became cured, therefore RNodes ∈ U and S
- *PandemicTime* which represents the duration of the pandemic. As long as the pandemic persists, there will always be large-scale contamination
- R which is a random number associated with a node (person) and is used to talk about the reinfection of a node. Indeed, the World Health Organization (WHO) has had to define the probability of reinfection $P_{Reinfection}$ which is 14% (Martínez-Álvarez *et al*., 2020) and if $R < P_{Reinfection}$, the person is reinfected
- $P_{Propagation}$ which represents the probability of propagation and $P_{Propagation} \in W$

Function Infection

This function manages four lists: Nodes at risk of infection, infected nodes (the current set of infected individuals), uninfected nodes, nodes that have been cured and newly infected nodes (the set of newly infected nodes generated by the spread of coronavirus from currently infected nodes). We are looking for the sub-graph of infected nodes at the output.

Initially, the subgraph of infected nodes is associated with the empty set $I = (V', E', W) \leftarrow \theta$ where V' is the set of infected nodes and E is the border set associated with its infected nodes. The vertex i^{th} of the randomly selected graph is considered the initial vertex $v_i \leftarrow v_{initial}$ and is added to the empty infected node set defined at the beginning $V' \leftarrow V' \cup \{v_i\}$, i.e., it is the infected node that allows the infection function to start. The number of iterations is controlled by the main loop, which evaluates the duration of the pandemic because as long as the pandemic is not over, the loop will continue to run but also with all infected nodes strictly greater than 0, meaning that there is at least one infected person (*time* < *PandemicTime and SizeOf* V' >0). Indeed, at least one person would have to be infected for the virus to spread. We then check the condition for the initial node $v_i \in V$, such that if its degree is strictly greater than zero ($d_G(v_i)$ >0) and the probabilities of propagation with its direct neighbors are strictly greater than zero ($P_{Propagation}(v_i,v_{i+1})$ >0), then we get a list of nodes that will be nodes infected by the initial node (*Infected Nodes*). The condition $d_G(v_i)$ >0 is checked because it can be in the case where the number of relations of a node with its neighbors is zero and it is said that it is an isolated person or a dead person so not in the presence of a virus spread.

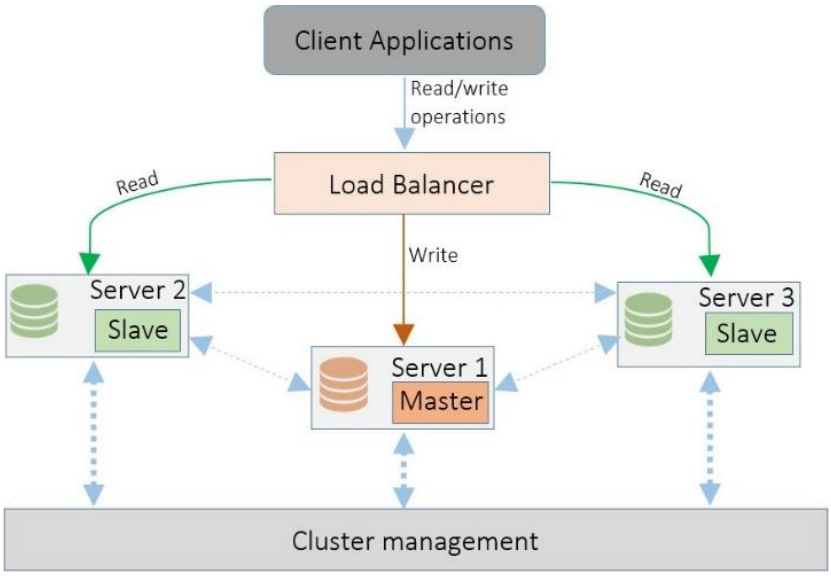

Fig. 5: Neo4j cluster architecture

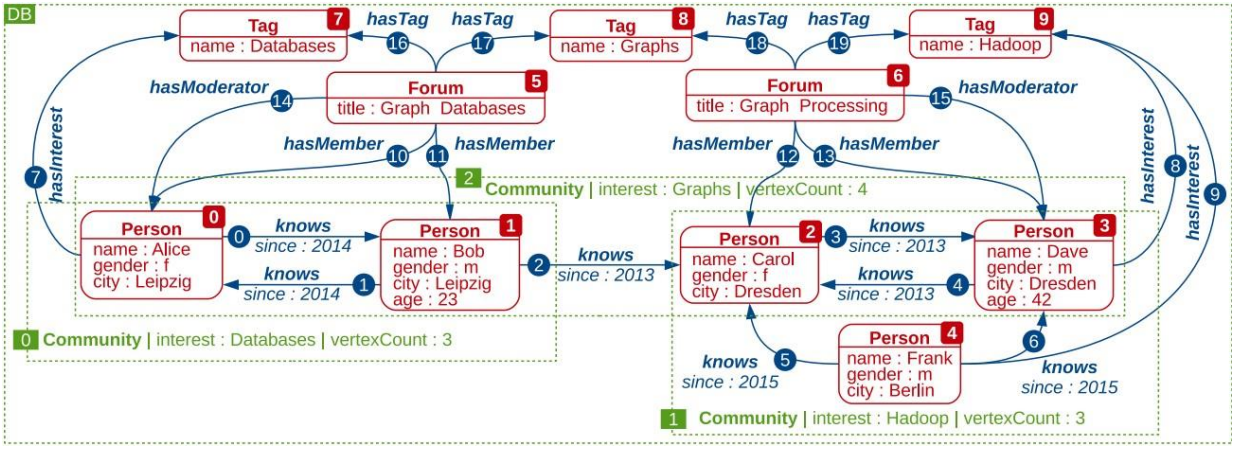

Fig. 6: Example EPGM database representing a simple social network containing three logical graphs (Junghanns *et al*., 2016)

After checking that the previously obtained list of infected nodes is not empty, the list of infected nodes is sent directly to a new list called the list of newly infected nodes (*NodesNewlyInfected* ← *InfectedNodes*). We then send the list of newly infected nodes obtained in the set of infected nodes (*V* ′ ← *NodesNewlyInfected*). We therefore say that this set of infected nodes is completed as the pandemic continues to exist, hence the condition time <*PandemicTime* and the increment *time* ← *time* +1 have been placed in our algorithm to signify the existence of this pandemic. Neighbors of the infected node include both infected nodes, uninfected nodes, cured nodes and nodes at risk of infection connected by edges defined by the probabilities of virus propagation, providing a broad means of observing the propagation of the virus in a community graph.

Function Reinfection

This function is used to treat the case of the new infection. We say that for the node belonging to the list of infected nodes ($v \in$ *InfectedNodes*) therefore is not a cured node and is not a susceptible node. However, if the node is cured, it can still be reinfected if its probability of reinfection is less than the value defined by WHO ($R<P_{Reinfection}$).

Function COV19-Dijkstra

Indeed, during propagation, from the initial infected node, a shorter path can be obtained and a prediction of the possible infected node can be made in a graph. As explained above with the Dijkstra algorithm, two sets are interesting, namely the set V which contains all the vertices of a community network and the set V' which contains the promising vertices. All the vertices have as structure the key-value pair according to the EPGM model seen above.

The first step is to initialize the distance $dist(v_{initial})$ to 0 ($dist(v_{initial}) \leftarrow 0$) and to initially assume that all the vertices of the graph $v_i \in V$ have as infinite distance ($dist(v_i) \leftarrow \infty$). We take the node i^{th} of the graph as the initial node ($v_i \leftarrow v_{initial}$). In the loop $V' \neq V$ which ensures the functioning of the algorithm, we start by extracting the minimum distance which is for the first iteration v_i that we will note $v_{current}$ and, which is added in the set of infected nodes ($V \cup \{v_{current}\}$) which is empty at the beginning ($V' \leftarrow \theta$). The next step thus consists in exploring and selecting the promising vertices until all the vertices of the target graph are found using the function add-new-infected-node and then we use the function Generate-Path to extract the path of the selected vertices and a concatenation is made to constitute all the shortest paths from the initial node to all the other nodes of the graph.

Algorithm 1: Function infection

Input
$G = (V, E, W)$: Undirected Graph
$v_{initial}$: Node initial
Output
$I = (V', E', W)$: Subgraph of infected nodes
Start
$I = (V', E', W) \leftarrow \theta$
time ← 0
$v_i \leftarrow v_{initial}$
$V' \leftarrow V' \cup \{v_i\}$
while *time* < *PandemicTime and SizeOf*(V') > 0 **do**
  for $v_i \in V'$ **do**
    if $d_G(v_i) > 0$ *and* $\mathbf{P}_{Propagation}(v_i, v_{i+1}) > 0$ **then**
      In f ectedNodes ← infect(SNode, infected, RNodes, $\mathbf{P}_{Propagation}$)
      if $N_{InfectedNodes} > 0$ **then**
        NodesNewlyIn fected ←*InfectedNodes*
      end
    end
  end
  $V' \leftarrow$ *NodesNewlyInfected*
  time ← *time* + 1
end

Algorithm 2: Function reinfection

Input
G = (*V, E, W*): Undirected Graph
Output
NewInfectedNodes
Start
R ← RandomNumber()
for *v* ∈ *InfectedNodes* **do**
  if v∉ *SNodes* **then**
    if v ∉ *RNodes* **then**
      NewInfectedNodes ← v
    else
      if R < $P_{Reinfection}$ **then**
        NewInfectedNodes ← v
        remove v from RNodes
      end
    end
  end
end

Algorithm 3: Function Cov19-Dijkstra

Input
G = (*V, E, W*): Graph
$v_{initial}$: Node initial
Output
GP: All the shortest paths
Start
V ′ ← θ
dist($v_{initial}$) ← 0
for ***all*** v_i ∈ *V* **do**
  dist(vi) ← ∞
end
v_i ← $v_{initial}$
while *V* ′/= *V* **do**
  $v_{current}$ ← Choose the minimum distance between all vertices
  V ′ ← *V* ′ ∪ {vcurrent }
  V ′ ← Add-New-Infected-Node(*G*, $v_{current}$)
  E′ ← Generate-Path(*V* ′, $v_{current}$)
end

Algorithm 4 describes the function add-new-infected-node and takes as input the graph *G* and the current peak $v_{current}$ to be expanded. It then explores in depth the neighborhood of the current vertex. For each developed vertex, it evaluates whether this vertex is infected, at risk of infection, not infected or already healed. It thus evaluates if the condition $dist(v_{current})+ w(v_{current}, v_{i+1}) < dist(v_{i+1})$ is checked, then $dist(v_{i+1}) \leftarrow dist(v_{current})+ w(v_{current}, v_{i+1})$. The node v_{i+1} thus becomes infected and we add it to the set of infected nodes *V* ′ (*V* ′ ← *V* ′ ∪{v_{i+1}}).

Algorithm 5 describes the function Generate-Path ′ and takes as input the set of infected nodes *V*′ produced in the add-new-infected-node function but also the current vertex $v_{current}$. Knowing that we have $v_{current}$ to reach v_{i+1}, which represents the edge {($v_{current}$, v_{i+1})} and will be added to the set of edges of the infected nodes sub-graph *E*′ ← *E*′ ∪ {($v_{current}$, v_{i+1})} which is empty at the beginning *E* ′ ← θ. Finally, we concatenate the edges of ′ our edges which are the vertices contained in *V* in order to build the paths.

Algorithm 4: Function add-new-infected-node

Input
G = (*V, E, W*): Graph
$v_{current}$: Infected start vertex
Output
V ′: Set of infected nodes
Start
for ***each*** v_{i+1} ∈ *V* and ($v_{current}$, v_{i+1}) ∈ *E* such that $\mathbf{P}_{Propagation}$ ($v_{current}$, v_{i+1}) > 0 **do**
  if *dist* ($v_{current}$) + *w*($v_{current}$, v_{i+1}) < *dist*(v_{i+1}) **then**
    dist(v_{i+1}) ← *dist*($v_{current}$) + *w*(*vcurrent*, v_{i+1})
    v_{i+1} ← Infected (SNodes, infected, RNodes, $\mathbf{P}_{Propagation}$)
    V ′← *V* ′ ∪ {v_{i+1}}
  end
end

Algorithm 5: Function generate-path

Input
V ′: Set of infected nodes
v_i: Current node
Output
GP: Generate the path
Start
E′← θ
for ***each*** v_{i+1} ∈ *V* ′ **do**
  E′← *E*′ ∪ {($v_{current}$, v_{i+1})}
  GP ← *Concat* (*E*′)
end

The output of the model, thus gives the sub-graph of infected nodes *I* = (*V* ', *E* ', *W*) which covers all the shortest paths from a source infected node to the other nodes that have become infected in the graph. While the algorithm is running, it evaluates the shortest paths from the infected node with the other nodes in the graph, but also helps to predict which nodes are at risk of infection and then take appropriate action to try to stop the chain of transmission.

Experimental Results

In this section, we present the results obtained by the COV-19-Dijkstra model. We will talk about the complexity of COV-19-Dijkstra computation time and propagation rate to know on average how many nodes an infected node can contaminate and present the different numbers of infected nodes obtained during the different iterations.

Environment Setup and Datasets

We used neo4j-community version 4.1.1 and an Apache NetBeans IDE 12.1 with Java 1.8.0 265; OpenJDK 64-Bit Server VM 25.265-b01 and OpenJDK Runtime Environment 1.8.0 265-8u265-b01-0ubuntu2 18.04-b01 under Linux system version 4. 15.0-117-generic running on amd64; UTF-8; in GB (nb) on a computer with Intel Core i5-8250U processor at 1.60 GHz × 8 and 7.7 GB RAM, running on Ubuntu 18.04.4 LTS operating system.

We perform our experimental studies using randomly generated data sets with the Neo4j benchmark randomized graph because of the need for hospitals and other medical services to keep their patient data private. The generator creates heterogeneous social network graphs with a fixed model similar to Fig. 6 where the nodes are Person type and the edges are Knowledge type with some probability of virus propagation on the edges that are randomly assigned. The generated graphs thus mimic the structural characteristics of real-world networks based on communities already known a priori (Ghasemian *et al.*, 2019). In generating these graphs, we have taken into account people with no social life, i.e., having no relationship with other people in the community or having only one relationship in the graph. In order to generate these graphs, a parameter μ which is the average fraction of nodes neighboring a node that does not belong to any community to which the reference node belongs, allowed us to control the fraction of edges between communities. In our case, μ = 0 to mean that all links are community links. Since the size of the communities was taken into account, we were able to generate graphs with many different nodes and at a large scale to better evaluate our model.

Time Complexity

In this section, we talk about the impact of increasing graph nodes on computation time. Indeed, we have carried out our study on graphs generated with 5000, 8000, 10000, 20000, 50000, 100000, 500000, 800000, 900000 and 1000000 nodes. Figure 7 shows the influence of the number of nodes on the computation time. Indeed, we say that the addition of new nodes promotes an exponential increase in temporal complexity.

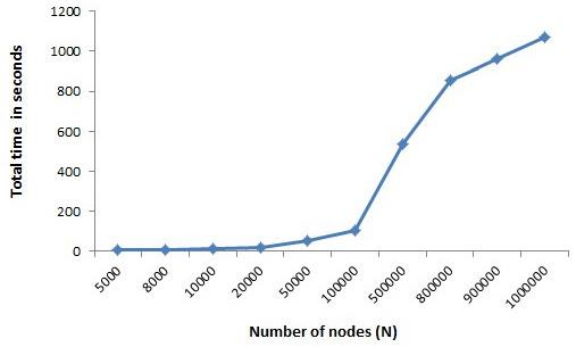

Fig. 7: Influence of number of nodes on the computation time

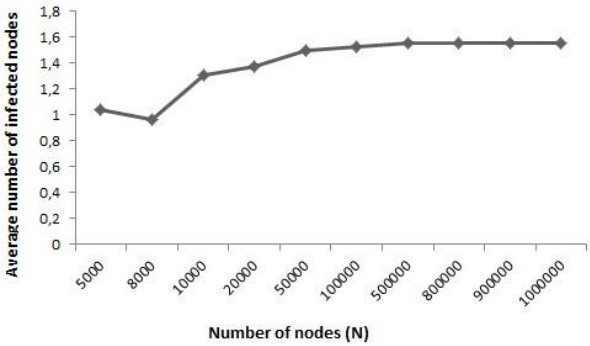

Fig. 8: Average number of nodes infected by a single infected node

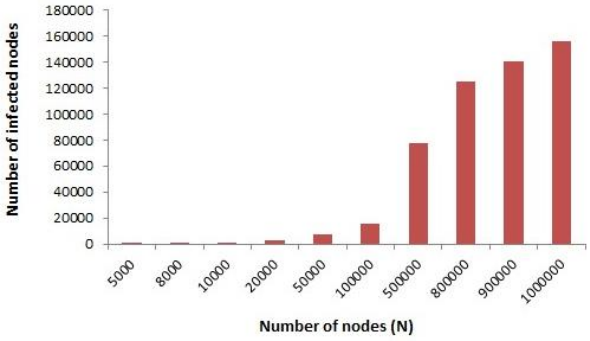

Fig. 9: The number of nodes contaminated by an infected node in the graph

We have noticed that the computing time will increase linearly each time the number of nodes increases. For example, we can see that with 100,000 nodes, the calculation time is between 0 and 200 sec, but multiplying this number by 100, we end up with a calculation time between 1000 and 1200 sec and even if we multiply the 100,000 nodes by half of 100, we find ourselves with a calculation time of between 400 and 600 sec.

The Propagation Rate

In this section, we present the results of the propagation rate, i.e., the number of nodes that are infected on average by an infected node in the graph (reproduction number). Figure 8 shows the result of this propagation rate from the number of infected nodes for the different numbers of graphs generated.

We can see that when comparing the propagation rate of a 5000 node graph to an 8000 node graph, there is a decrease in propagation rate. By further increasing the number of nodes, the propagation rate increases and at a generated graph of 500,000 nodes, the propagation rate seems to stabilize. This stability is due to the structure of the randomly generated graphs and to the good performance of our algorithm over a large number of nodes.

Figure 9 shows us the number of infected nodes obtained for the different graphs generated. We say that

the result obtained is normal because the number of nodes infected by an infected node in the graph will continue to increase as the number of nodes in a graph increases. We notice that the more the number of nodes in the generated graphs increases, the more the number of infected nodes obtained for the different generated graphs increases. When we try to compare the number of infected nodes with the computation time versus the increase in the number of nodes, we say that as the computation time increases, so does the number of infected nodes. Therefore, the number of infected nodes depends on the computation time of our algorithm which depends on the number of nodes in the generated graphs.

Discussion

In this section, we discuss the aspects and advantages of our COV-19-Dijkstra: COVID-19 propagation model based on Dijkstra's algorithm. First, one aspect concerns the optimal use of the graph. Indeed, we can say that a graph is optimal when all the nodes of the graph participate in the computation of the complete path. The answer to this question depends on the configuration of the graph that is generated. Being in a real-world configuration, there must be nodes that are not linked to other nodes. We are talking about cases of isolation and quarantine. That is why in our virus propagation model we specified the point where $d_G(v_i) > 0$ as a solution to this problem.

The second aspect concerns the level of spread of the virus. Indeed, it is difficult to see the case when two nodes that are linked have a zero probability of propagation. To remedy this, in our propagation algorithm we thought of the following condition $P_{Prop}(v_{current}, v_{i+1}) > 0$.

Finally, the third aspect is the direct neighbor of an infected node. We may have a node (person) that already respects the barrier measures, so it is difficult to say that this node will be infected. Indeed, the idea of this study is to be able to predict which nodes are at risk of infection and to mitigate the chain of transmission of COVID-19, while not neglecting any case.

Conclusion and Future Work

To summarize, we proposed in this study a COVID-19 propagation model based on the Dijkstra algorithm called COV19-Dijkstra to be able to compute the shortest path of COVID-19 infection in large-scale community networks in order to analyze and predict the transmission chain. Experiments have shown that our COV19-Dijkstra model is very powerful, very scalable and works well on large-scale graphs. Moreover, it is reliable and can be used in the fight against the rapid spread of COVID-19 in low and high population density areas. Thus, we say that our work represents a contribution to the fight against Covid-19 and could generate more interest for other shorter path algorithms in graphs or for other infectious diseases.

For future work, we are interested in:

- The use of such a model on information transmission networks such as twitter concerning an infectious disease using the big data graph like Neo4j
- The proposal of a new geospatial data model using the A* algorithm to plot and analyze the spread of the virus in a graph
- The use of such a model which is proposed on a decentralized database for the storage and transmission of information such as the blockchain

Acknowledgment

Thank you to the publisher for their support in the publication of this research article. We are grateful for the resources and platform provided by the publisher, which have enabled us to share our findings with a wider audience. We appreciate the efforts of the editorial team in reviewing and editing our work, and we are thankful for the opportunity to contribute to the field of research through this publication.

Funding Information

The authors have not received any financial support or funding to report.

Author's Contributions

Arnaud Watusadisi Mavakala and Wilfried Yves Hamilton Adoni: Developed the theory and performed the computations.

Najib Ben Aoun: Helps in the implementation part as well as design.

Tarik Nahhal: Conceived the presented idea.

Moez Krichen: Conduct literature review and perform the analysis.

Mohammed Y. Alzahrani: Provide first paper draft and interpretated the results.

Franck Mutombo Kalala: Supervise all the work, validate the idea.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Al Hakim, R. R., Purwono, P., Arief, Y. Z., Pangestu, A., Satria, M. H., & Ariyanto, E. (2022). Implementation of Dijkstra Algorithm with React Native to Determine COVID-19 Distribution. *Sistemasi: Jurnal Sistem Informasi*, *11*(1), 160-170. http://sistemasi.ftik.unisi.ac.id/index.php/stmsi/article/view/1667

Alguliyev, R., Aliguliyev, R., & Yusifov, F. (2021). Graph modelling for tracking the COVID-19 pandemic spread. *Infectious disease modelling*, *6*, 112-122. https://doi.org/10.1016/j.idm.2020.12.002

Andersen, K. G., Rambaut, A., Lipkin, W. I., Holmes, E. C., & Garry, R. F. (2020). The proximal origin of SARS-CoV-2. *Nature medicine*, *26*(4), 450-452. https://doi.org/10.1038/s41591-020-0820-9

Aoun, N. B., Elghazel, H., Hacid, M. S., & Amar, C. B. (2011a, August). Graph aggregation based image modeling and indexing for video annotation. In *International Conference on Computer Analysis of Images and Patterns* (pp. 324-331). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-23678-5_38

Aoun, N. B., Elghazel, H., & Amar, C. B. (2011b, April). Graph modeling based video event detection. In *2011 International Conference on Innovations in Information Technology* (pp. 114-117). IEEE. https://ieeexplore.ieee.org/abstract/document/5893799

Aoun, N. B., Mejdoub, M., & Amar, C. B. (2014, May). Bag of sub-graphs for video event recognition. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 1547-1551). IEEE. https://ieeexplore.ieee.org/abstract/document/6853857

Block, P., Hoffman, M., Raabe, I. J., Dowd, J. B., Rahal, C., Kashyap, R., & Mills, M. C. (2020). Social network-based distancing strategies to flatten the COVID-19 curve in a post-lockdown world. *Nature Human Behaviour*, *4*(6), 588-596. https://doi.org/10.1038/s41562-020-0898-6

Cai, F., Qingfeng, H., LanSheng, H., Li, S., & Xiao-Yang, L. (2013). Virus propagation power of the dynamic network. *EURASIP Journal on Wireless Communications and Networking*, *2013*(1), 1-14. https://doi.org/10.1186/1687-1499-2013-210

Chan, J. F. W., Yuan, S., Kok, K. H., To, K. K. W., Chu, H., Yang, J., ... & Yuen, K. Y. (2020). A familial cluster of pneumonia associated with the 2019 novel coronavirus indicating person-to-person transmission: A study of a family cluster. *The lancet*, *395*(10223), 514-523. https://doi.org/10.1016/S0140-6736(20)30154-9

Chen, Y. C., Lu, P. E., Chang, C. S., & Liu, T. H. (2020). A time-dependent SIR model for COVID-19 with undetectable infected persons. *Ieee transactions on Network Science and Engineering*, *7*(4), 3279-3294. https://ieeexplore.ieee.org/abstract/document/9200529

de Bruin, Y. B., Lequarre, A. S., McCourt, J., Clevestig, P., Pigazzani, F., Jeddi, M. Z., ... & Goulart, M. (2020). Initial impacts of global risk mitigation measures taken during the combatting of the COVID-19 pandemic. *Safety Science*, *128*, 104773. https://doi.org/10.1016/j.ssci.2020.104773

Faranda, D., & Alberti, T. (2020). Modeling the second wave of COVID-19 infections in France and Italy via a stochastic SEIR model. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *30*(11), 111101. https://doi.org/10.1063/5.0015943

Ghasemian, A., Hosseinmardi, H., & Clauset, A. (2019). Evaluating overfit and underfit in models of network community structure. *IEEE Transactions on Knowledge and Data Engineering*, *32*(9), 1722-1735. https://ieeexplore.ieee.org/abstract/document/8692626

Giannis, D., Ziogas, I. A., & Gianni, P. (2020). Coagulation disorders in coronavirus infected patients: COVID-19, SARS-CoV-1, MERS-CoV and lessons from the past. *Journal of Clinical Virology*, *127*, 104362. https://doi.org/10.1016/j.jcv.2020.104362

Hope, T., Portenoy, J., Vasan, K., Borchardt, J., Horvitz, E., Weld, D. S., ... & West, J. (2020). SciSight: Combining faceted navigation and research group detection for COVID-19 exploratory scientific search. *arXiv preprint arXiv:2005.12668*. https://arxiv.org/abs/2005.12668

Jabri, E. (2020). A Novel Approach for Generating SPARQL Queries from RDF Graphs. *arXiv preprint arXiv:2006.02862*. https://arxiv.org/abs/2006.02862

Junghanns, M., Petermann, A., Teichmann, N., Gómez, K., & Rahm, E. (2016, July). Analyzing extended property graphs with Apache Flink. In *Proceedings of the 1st ACM SIGMOD Workshop on Network Data Analytics* (pp. 1-8). https://doi.org/10.1145/2980523.2980527

Kan, B., Zhu, W., Liu, G., Chen, X., Shi, D., & Yu, W. (2017). Topology modeling and analysis of a power grid network using a graph database. *International Journal of Computational Intelligence Systems*, *10*(1), 1355-1363. https://doi.org/10.2991/ijcis.10.1.96

Khashan, E. A., Eldesouky, A. I., Fadel, M., & Elghamrawy, S. M. (2020). A big data based framework for executing complex query over Covid-19 datasets (Covid-QF). *arXiv preprint arXiv:2005.12271*. https://arxiv.org/abs/2005.12271

Lewnard, J. A., & Lo, N. C. (2020). Scientific and ethical basis for social-distancing interventions against COVID-19. *The Lancet Infectious Diseases*, *20*(6), 631-633. https://www.thelancet.com/journals/lancet/article/PIIS1473-3099(20)30190-0/fulltext

Liang, K. (2020). Mathematical model of infection kinetics and its analysis for COVID-19, SARS and MERS. *Infection, Genetics and Evolution*, *82*, 104306. https://doi.org/10.1016/j.meegid.2020.104306

Lin, S., Huang, J., He, Z., & Zhan, D. (2020). Which measures are effective in containing covid-19?—empirical research based on prevention and control cases in China. *MedRxiv*. https://doi.org/10.1101/2020.03.28.20046110

Martínez-Álvarez, F., Asencio-Cortés, G., Torres, J. F., Gutiérrez-Avilés, D., Melgar-García, L., Pérez-Chacón, R., ... & Troncoso, A. (2020). Coronavirus optimization algorithm: A bioinspired metaheuristic based on the COVID-19 propagation model. *Big Data*, *8*(4), 308-322. https://doi.org/10.1089/big.2020.0051

Mejdoub, M., Aoun, N. B., & Amar, C. B. (2015). Bag of frequent subgraphs approach for image classification. *Intelligent Data Analysis*, *19*(1), 75-88. https://content.iospress.com/articles/intelligent-data-analysis/ida00697

Needham, M., & Hodler, A. E. (2019). *Graph algorithms: Practical examples in Apache Spark and Neo4j*. O'Reilly Media.

Ordun, C., Purushotham, S., & Raff, E. (2020). Exploratory analysis of covid-19 tweets using topic modeling, umap and digraphs. *arXiv preprint arXiv:2005.03082*. https://arxiv.org/abs/2005.03082

Rădulescu, A., Williams, C., & Cavanagh, K. (2020). Management strategies in a SEIR-type model of COVID 19 community spread. *Scientific reports*, *10*(1), 1-16. https://doi.org/10.1038/s41598-020-77628-4

Rodriguez, M. A., & Neubauer, P. (2010). Constructions from dots and lines. *arXiv preprint arXiv:1006.2361*. https://arxiv.org/abs/1006.2361

Ru, H., Yang, E., & Zou, K. (2020). What do we learn from SARS-CoV-1 to SARS-CoV-2: Evidence from global stock markets. *Available at SSRN*, *3569330*.

Sahin, A. R., Erdogan, A., Agaoglu, P. M., Dineri, Y., Cakirci, A. Y., Senel, M. E., ... & Tasdogan, A. M. (2020). 2019 novel coronavirus (COVID-19) outbreak: A review of the current literature. *EJMO*, *4*(1), 1-7.

Song, B., & Hei, X. (2020). Models and strategies on reopening lockdown societies due to COVID-19. *OSF Prepr*, *10*. https://www.xialihei.com/wp-content/uploads/2020/04/English.pdf

Sameni, R. (2020). Mathematical modeling of epidemic diseases; a case study of the COVID-19 coronavirus. *arXiv preprint arXiv:2003.11371*. https://arxiv.org/abs/2003.11371

Sebastiani, G., Massa, M., & Riboli, E. (2020). COVID-19 epidemic in Italy: Evolution, projections and impact of government measures. *European Journal of Epidemiology*, *35*(4), 341-345. https://doi.org/10.1007/s10654-020-00631-6

Silveira, P., Teixeira, A. P., & Soares, C. G. (2022). Characterisation of ship routes off the continental coast of Portugal using the Dijkstra algorithm. *Trends in Maritime Technology and Engineering Volume 2*, 151-159. ISBN-10: 9781003320289.

Yang, C., & Wang, J. (2020). A mathematical model for the novel coronavirus epidemic in Wuhan, China. *Mathematical Biosciences and Engineering: MBE*, *17*(3), 2708. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7376496/

Yang, Y., Li, Z., Wang, X., & Hu, Q. (2019). Finding the shortest path with vertex constraint over large graphs. *Complexity*, *2019*. https://doi.org/10.1155/2019/8728245

Yilmaz, S., Dudkina, E., Bin, M., Crisostomi, E., Ferraro, P., Murray-Smith, R., ... & Shorten, R. (2020). Kemeny-based testing for COVID-19. *PLOS One*, *15*(11), e0242401. https://doi.org/10.1371/journal.pone.0242401

Zhu, L., Zhang, Y. H., Su, F., Chen, L., Huang, T., & Cai, Y. D. (2016). A shortest-path-based method for the analysis and prediction of fruit-related genes in arabidopsis thaliana. *PloS one*, *11*(7), e0159519. https://doi.org/10.1371/journal.pone.0159519

Zorzenon, D., Molinari, F., & Raisch, J. (2021, May). Low Complexity Method for Simulation of Epidemics Based on Dijkstra's Algorithm. In *2021 American Control Conference (ACC)* (pp. 3018-3025). IEEE. https://ieeexplore.ieee.org/abstract/document/9483311