ISSN 2347 - 3983
Volume 8. No. 7, July 2020
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter09872020.pdf
https://doi.org/10.30534/ijeter/2020/09872020

A New Algorithm for Contact Trace Network Evolution and Visualization

Monday Eze[1], Chigozirim Ajaegbu[2], Olusola Maitanmi[3], Doris Nnakwuzie[4]
[1]Department of Computer Science, Babcock University, Nigeria. ezem@babcock.edu.ng
[2]Department of Computer Science, Babcock University, Nigeria. ajaegbuc@babcock.edu.ng
[3]Department of Software Engineering, Babcock University, Nigeria. maitanmio@babcock.edu.ng
[4]Dept. of Computer Science, Alex Ekwueme Federal University, Nigeria. okafordoris49@yahoo.com

ABSTRACT

The necessity for contact tracing in the fight against infectious diseases including pandemics like COVID 19 cannot be overemphasized. One of the obvious challenges is how to device practical strategies for computational evolution and construction of a homogenous network for contact tracing. A second challenge is how to evolve practical tools and algorithms for the visualization of contact network that models transmission. This research evolves a new algorithm which first, builds a new specialized data structure known as PiVector, and then a corresponding contact network visualization system. The resulting network could be used for contact tracing of the transmission of infectious diseases. This work practically demonstrates a programmatic and data-driven approach to network evolution, construction and visualization. This work was implemented using Python Programing Language and other related data mining technologies.

Key words: Homogenous, Contact Network, PiVector Construction, Visualization, Python.

1. INTRODUCTION

Network Theory has found its application in several research fields as well as diverse areas of human endeavours [1]. Some of the application areas are social networks, biological networks, semantic networks used in computational linguistics, cognitive networks in neurosciences, transportation networks, and telecommunication networks, among others. Literature has shown that the evolutionary foundation of all these networks is the mathematical graph [2]. A graph is defined as a collection of vertices, which are connected to each other through edges. In other words, each edge of graph joins two vertices [3]. A network is an instantiation of a graph, such that the nodes and edges have specific identities. For instance, a social network comprises of nodes as human beings, while the edges are the links or relationships between those human beings [4]. Again, a road network comprises of nodes as cities, while the edges are the actual roads linking the cities [5]. Research has underlined the necessity for efficient algorithms and programmatic techniques for generating complex networks [6]. This current work demonstrates how to achieve this goal, in a procedural manner, beginning from the workflow stage, to the implementation of the proposed algorithm.

Infectious diseases spread through human contacts [7], which could be modelled as a network. In order to control the spread of such diseases, it is very necessary to successfully locate the infected persons, and all the yet-to-be-infected contacts, so they could be isolated or treated as the case may be. This activity is known as contact tracing [8]. The necessity for contact tracing and its automation in the fight against pandemics like COVID 19 [9] cannot be overemphasized. Epidemiologists [10] cannot afford to depend on manual contact tracing. Thus the need for this research, which is geared at developing a new computational algorithm [11] for network evolution and visualization. Tackling the challenges of devising practical strategies for computational evolution and construction of a homogenous network for contact tracing are the major deliverables of this work. This research evolves a new specialized data structure known as PiVector, then uses it as input to construct the network for use in contact tracing of infectious diseases transmission.

2. EVOLUTIONARY CONCEPTS

This section presents some evolutionary concepts [12] which constitute the foundation of this research. In order to effectively generate a computational network, it is necessary to propose a requisite data structure [13]. As shown in Figure 1, the spread of a pandemic such as COVID-19 begins with the establishment of a contact with an infected person (H1), and then a successful transmission from the infected case to another human being (H2).

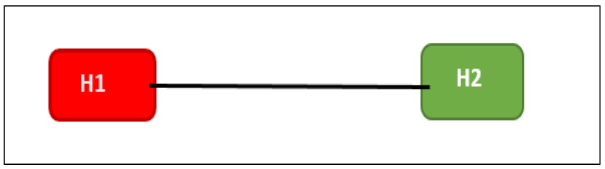

Figure 1: Evolutionary Diagram of a Contact Network

In other words, the contacts leading to disease transmission constitutes a homogenous contact network [14], with human nodes, and edges as the transient contacts between such human beings. Some of the deliverables of this work are a contact network data structure, an overall workflow, and a transmission network visualization which could be useful for contact tracing. As shown in the evolutionary diagram, colours could be used at the design level to designate infection status. For instance, the infected node H1 is coloured RED while the uninfected node H2 is GREEN.

3. SYSTEM WORKFLOW

The system workflow [15] for this research is in two parts. These are the PiVector Generation and the System Implementation workflows respectively. The first generates the PiVector, which is an input into the second workflow that drives the actual system implementation [16]. Both will be explained in details.

3.1 PiVector Generation Workflow

The PiVector Generation workflow is shown in Figure 2. As indicated in the figure, there are eight chronological steps, numbered from 1 to 8, necessary for generating the digitized dataset [17] called PiVector.

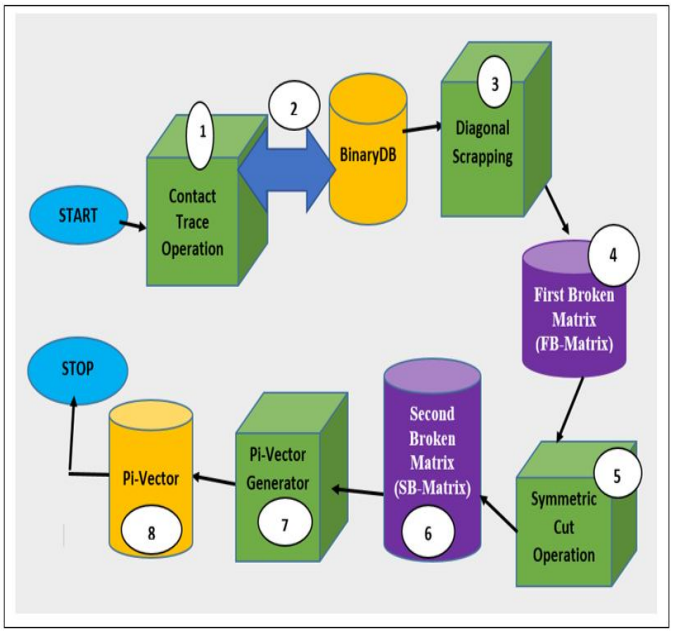

Figure 2: The PiVector Generation Workflow

The workflow begins with a manual or human intensive Contact Trace Operation. Just like any contact tracing [18] in epidemiological researches, this activity involves attempting to identity persons who have made contacts with infected individuals [19]. The outcome of contact tracing is transformed into a digital register known as binary database (BinaryDB). The evolution of BinaryDB from manual contact tracing operation is a bi-directional [20] transition shown in the workflow diagram using a two-way arrow. This is because, the human component of contact tracing and corresponding update of the binary data [21] is a continuous process, as long as the infection and corresponding network continue to exist. In its simplest form, the BinaryDB is a binary square matrix structure. For instance, the BinaryDB in Figure 3, used in this work consists of a 15 by 15 square matrix extracted through human contact tracing. The rows and columns are formed using the tuple {H1, H2, H3, H4, H5, ...H15}, which represent the human nodes involved in network contacts. It follows that for any integer x and y within the range of number of network nodes, if any two network nodes Hx and Hy were in contact, then their intersection in the BinaryDB is marked with value "1", else it is marked with value of "0". The third step in the workflow involves scrapping the TLBR (top left – bottom right) diagonal of the BinaryDB. Scrapping the TLBR diagonal is an optimization step taken to reduce unnecessary memory overheads [22].

	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12	H13	H14	H15
H1	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0
H2	1	1	1	1	0	0	1	0	1	0	0	1	0	0	0
H3	0	1	1	0	1	0	0	0	0	0	1	0	0	0	0
H4	0	1	0	1	0	1	0	1	0	0	1	0	0	0	0
H5	1	0	1	0	1	0	0	0	0	1	0	1	0	0	0
H6	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1
H7	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0
H8	0	0	0	1	0	0	1	1	0	0	1	0	1	0	0
H9	0	1	0	0	0	0	0	0	1	1	0	1	0	0	0
H10	0	0	0	0	1	0	0	0	1	1	0	0	0	1	0
H11	0	0	1	1	0	0	0	1	0	0	1	0	0	1	0
H12	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0
H13	0	0	0	0	0	1	0	1	0	0	0	0	1	0	1
H14	1	0	0	0	0	0	0	0	0	1	1	0	0	1	0
H15	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1

Figure 3: The BinaryDB for a 15x15 Contact Network

The reason for scrapping is because all the entries in the TLBR diagonal represent contacts between nodes [23] and themselves, such that the equation 1 always holds:

$$BinaryDB(H_x, H_y) = 1 \quad for\ two\ integers\ x, y\ where\ x = y \qquad (1)$$

The outcome of the scrapping operation is that the entire contents of the TLBR diagonal [24] are changed from "1" to

"*" as shown in Figure 4, a result which is termed the FB-Matrix (First Broken Matrix). The next step in the workflow is the formation of SB-Matrix (Second Broken Matrix). This is achieved through a Symmetric Cut Operation which discards the Lower Left side of the BinaryDB. The reason is that a typical BinaryDB is symmetric [25], and as such the following equation always holds:

$$BinaryDB(H_x, H_y) = BinaryDB(H_y, H_x) \quad (2)$$

for every H_x, H_y, and integers x and y.

Two illustrations of the concept of symmetric equivalence have been clearly shown in Figure 4 by circling the entries H4xH8 and H8xH4 respectively in the BinaryDB.

	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12	H13	H14	H15
H1	*	1	0	0	1	0	0	0	0	0	0	0	0	1	0
H2	1	*	1	1	0	0	1	0	1	0	0	1	0	0	0
H3	0	1	*	0	1	0	0	0	0	0	1	0	0	0	0
H4	0	1	0	*	0	1	0	1	0	0	1	0	0	0	0
H5	1	0	1	0	*	0	0	0	0	1	0	1	0	0	0
H6	0	0	0	1	0	*	0	0	0	0	0	0	1	0	1
H7	0	1	0	0	0	0	*	1	0	0	0	0	0	0	0
H8	0	0	0	1	0	0	1	*	0	0	1	0	1	0	0
H9	0	1	0	0	0	0	0	0	*	1	0	1	0	0	0
H10	0	0	0	0	1	0	0	0	1	*	0	0	0	1	0
H11	0	0	1	1	0	0	0	1	0	0	*	0	0	1	0
H12	0	1	0	0	1	0	0	0	1	0	0	*	0	0	0
H13	0	0	0	0	0	1	0	1	0	0	0	0	*	0	1
H14	1	0	0	0	0	0	0	0	0	1	1	0	0	*	0
H15	0	0	0	0	0	1	0	0	0	0	0	0	0	0	*

Figure 4: A Sample 15 x 15 FB-Matrix

The SB-Matrix is shown in Figure 5. The SB-Matrix is used as input to the Pi-Vector Generator, which gives rise to the PiVector, a special optimized data structure [26] used as final input for the actual construction of the contact network.

	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12	H13	H14	H15
H1	*	1	0	0	1	0	0	0	0	0	0	0	0	1	0
H2	*	*	1	1	0	0	1	0	1	0	0	1	0	0	0
H3	*	*	*	0	1	0	0	0	0	0	1	0	0	0	0
H4	*	*	*	*	0	1	0	1	0	0	1	0	0	0	0
H5	*	*	*	*	*	0	0	0	0	1	0	1	0	0	0
H6	*	*	*	*	*	*	0	0	0	0	0	0	1	0	1
H7	*	*	*	*	*	*	*	1	0	0	0	0	0	0	0
H8	*	*	*	*	*	*	*	*	0	0	1	0	1	0	0
H9	*	*	*	*	*	*	*	*	*	1	0	1	0	0	0
H10	*	*	*	*	*	*	*	*	*	*	0	0	0	1	0
H11	*	*	*	*	*	*	*	*	*	*	*	0	0	1	0
H12	*	*	*	*	*	*	*	*	*	*	*	*	0	0	0
H13	*	*	*	*	*	*	*	*	*	*	*	*	*	0	1
H14	*	*	*	*	*	*	*	*	*	*	*	*	*	*	0
H15	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

Figure 5: A Sample 15 x 15 SB-Matrix

3.1.1 The PiVector Generation

It is important to discuss what PiVector is all about, how it is generated, and why it is very optimal. The term PiVector is derived from the word "Pipe" for symbol "|". The PiVector generator scans through the SB-Matrix, and selects all the entries that indicate contacts between nodes. These are the cell entries having binary values as "1". All such nodes are collated, and used to build a new row vector [27] such that the entries in the vector comprise of a concatenation of nodes using pipe symbols on a row by row basis, until the entire SB-Matrix entries are fully traversed [28]. In other words, the first two items in the PiVector will be PiVector (1) = "H1|H2" and PiVector(2)= "H1|H5" respectively. The resulting PiVector shown in Figure 6 is a row vector having 25 elements, generated from the full traversal of the SB-Matrix.

PiVector	H1|H2	H1|H5	H1|H14	H2|H3	H2|H4	H2|H7	H2|H9	H2|H12	H3|H5
Index	1	2	3	4	5	6	7	8	9

PiVector	H3|H11	H4|H6	H4|H8	H4|H11	H5|H10	H5|H12	H6|H13	H6|H15	H7|H8
Index	10	11	12	13	14	15	16	17	18

PiVector	H8|H11	H8|H13	H9|H10	H9|H12	H10|H14	H11|H14	H13|H15
Index	19	20	21	22	23	24	25

Figure 6: A Sample PiVector

3.1.2 Optimization Calculation

The use of PiVector as a data structure for storing binary data for network generation has a number of benefits, especially in the area of storage optimization [29]. The percentage gain in storage space can be calculated by comparing the size of the final dataset with the original BinaryDB, for instance:

Storage space of PiVector = 25 (the count of number of rows).
Storage space of BinaryDB = 225 (that is 15 x15).
Storage Gain = (225-25) = 200
Optimization % = (200/225) * 100 = 88.9%

3.2 System Implementation Workflow

The detailed system implementation workflow of this research is shown in Figure 7. The programming language used is Python, though it could be re-implemented in other high level languages with minimal modifications. The system implementation workflow is numbered chronologically from Step 1 to Step 7 as shown. Step 1 is called the Node Colour Specification [30]. This involves the specification of the colours of the network nodes. As earlier mentioned and demonstrated in the evolutionary diagram in Figure 1, the two predefined colours for the network nodes in this research are RED for identifying the infected nodes, and GREEN for the rest of the nodes that have come in contact with the infected, but they themselves are yet to be infected.

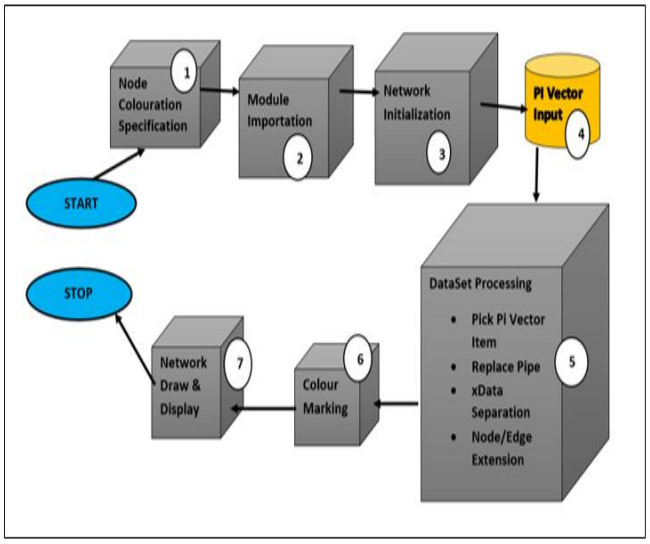

Figure 7: System Implementation Workflow

Step 2 which is the Module Importation constitutes the second step in the system implementation. The two major external modules that were brought into Python environment using the import statements are networkx [31] and matplotlib.pyplot [32] respectively. These two modules are key to graphics design and plotting in Python. After importing necessary external modules, Step 3 in the chronology of implementation is to initialize the network. This is achieved through a generic assignment statement shown in equation 3.

$$G = nx.Graph(\ ) \quad (3)$$

where G is the graph identifier, nx is an alias linked to networkx importation statement, and Graph () is a graph initialization function.

In Step 4, system accepts the PiVector, earlier generated in the first workflow. This is followed by Step 5, which is a series of loop controlled dataset processing operations [33]. For each of the items in the PiVector data structure, the pipe "|" symbol that is used to separate data items are first replaced with empty symbols (characters). For instance, a typical piped item such as [A|B] is transformed as shown in equation 4:

$$Replacement: [A \mid B] \rightarrow [A\ B] \quad (4)$$

where A and B are data items or identifiers, representing the nodes of a network under construction

The separated nodes such as A and B above are then used to populate the two item array called xData. The first item is pushed into xData [0] while the second one is pushed into xData [1] as shown in assignment statement in equation 5:

$$A \rightarrow xData[0], \quad B \rightarrow xData[1] \quad (5)$$

Thereafter, the contents of xData are used to extend the edges and the nodes of the contact network under construction. The system looping runs until all the contents of the PiVector are accessed and processed accordingly. Step 6 known as Colour Marking involves the actual marking of the nodal colours earlier specified. The source code for colouration for selected network nodes is shown in Figure 8.

```
# Colour Mapping for Network Nodes H1, H15, H8 and H12
redNodes =['H1', 'H15', 'H8', 'H12']
color_map = [ ] # Initialize a Colour Map before looping
   for node in G:
      if node in redNodes:
         color_map.append('red')
      else:
         color_map.append('green')
```

Figure 8: Node Colour Marking Code

The last computational step known as Network Draw and Display involves the actual drawing and visual display of the generated network object [34]. Network drawing and display is achieved using the lines of code in Figure 9. The network draw invocation call statement used three important arguments. These are the Graph Object, the node colour and the node size. On the other hands, the network display statement is a non-argument statement.

```
nx.draw_networkx(G, node_color=color_map, node_size=1000)
plt.show()
```

Figure 9: Network Drawing and Display Code

4. SYSTEM OUTPUT

The contact network generated as output of the system run is shown in Figure 10. As shown, there are a total of 15 human nodes under contact tracing. It can be deduced that a total of four nodes have already been infected by the disease. These are nodes H1, H8, H12 and H15. It is important to state that the node labelling in this research is very flexible. For instance, in this experimental run, the node labels were designated simplistically as Hx where x is an integer. However, this choice is not static, rather, it is possible to design node labels to take the exact names of the human nodes, example Alex, Daniel, Farad, Papa, and so on. It is also possible to use a form of real life unique identifications such as National ID or even Phone Numbers of the human nodes. In fact, car registration numbers [35] such as AB112X, YA655G, NU514P could also be used.

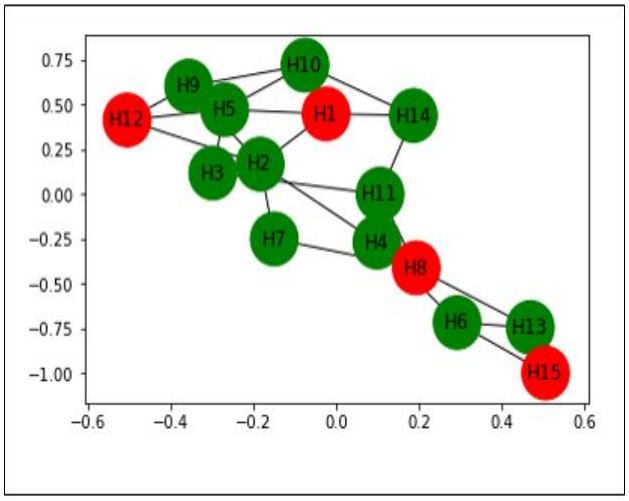

Figure 10: System Output

The system output forms a key visual resource for infectious disease contact tracing. For instance, it is very easy to deduce from the generated network that the human nodes H9, H5 and H3 are not yet infected, but have all come in contact with H12 which is currently the index case. Another important point is the symmetricity of the system runs and outputs. This implies that it is possible to generate multiple outcomes, which are symmetric in node arrangements, though the visual outlooks may appear different. In other words, rather than a single static output, the same network gets displayed in different graphical orientations [36] of nodes arrangements.

5. CONCLUSION

This research has demonstrated the implementation of contact networks evolution and visualization from the scratch. The outcome of this research could be a valuable tool in epidemiology research, especially in the area of contact tracing during pandemics such as COVID-19. The two major workflows used in this study were presented in unambiguous manner in such a way that other researchers could re-implement this work in future researches. The evolution and application of PiVector as a new data structure were explained, and mathematical calculation based on percentages was used to show how its usage could optimize storage space. Other deliverables of this work are the first broken matrix (FB-Matrix) and second broken matrix (SB-Matrix) respectively, whose importance and usages have been earlier explained. Future extension of this work will focus on inculcating of artificial intelligence-based search [37] into the process of contact tracing in a disease network.

REFERENCES

1. G. Rivera , **The use of Actor-Network Theory and a Practice-Based Approach to understand online community participation**, A PhD Thesis, Information School The University of Sheffield, May 2013
2. O. Mason and M. Verwoerd, **Graph Theory and Networks in Biology**, Hamilton Institute, National University of Ireland Maynooth, Co. Kildare, Ireland, January 17, 2007
3. DA. Costa, FA. Rodrigues, G. Travieso, and V. Boas, "Characterization of complex networks: A survey of measurements", Advances in Physics, Vol. 56, No. 1, Feb. 2007, 167–242 https://doi.org/10.1080/00018730601170527
4. P.B. Brandtzaeg, **Social Networking Sites: Their Users and Social Implications — A Longitudinal Study**, Journal of Computer-Mediated Communication 17(4):467–488. · August 2012
5. W. Krupowicz, K. Sobolewska, and M. Burinskiene, **Modern Trends in Road Network Development in Rural Areas**, Baltic Journal of Road and Bridge Engineering 12(1):48-56 · Feb. 2017 https://doi.org/10.3846/bjrbe.2017.06
6. M. J. McGuffin, **Simple Algorithms for Network Visualization: A Tutorial**, Tsingua Science and Technology, Vol. 17, No. 4, August 2012, pp1-16
7. L. Steele, E. Orefuwa and P. Dickmann. **Drivers of earlier infectious disease outbreak detection: a systematic literature review**. International Journal of Infectious Diseases, Vol 53, 2016, 15-20 https://doi.org/10.1016/j.ijid.2016.10.005
8. A. Greiner, K. Angelo, A. McCollum, K. Mirkovic, R. Arthur and F. Angulo. **Addressing contact tracing Challenges – critical to halting Ebola virus disease transmission**. International Journal of Infectious Diseases 41, 2015, 53-55
9. M. Adnan, S. Khan, A. Kazmi, N. Bashir and R. Siddque. **Covid 19 Infection: Origin, transmission and characteristics of human coronaviruses**. Journal of Advanced Research Vol 24, July 2020, 91-98.
10. J. Goncalves, E. Duarte, L. Jensen and L. Posenato. **Epidemiology and Health Services: 25 years in review**. Epidemiol. Serv. Saude, Brasilia, Vol. 24 (4), 2017, 1-16.
11. P. Vinista and M.M. Joe. **A Novel Modified Sobel Algorithm for Better Edge Detection of Various Images.** International Journal of Emerging Trends in Engineering Research (IJETER). Vol. 7, Issue 3, March 2019, 25-31
12. B. Francesco. **Evolutionary Computation Methods and their Applications in Statistics**. Statistica. 67, 2013, 201-224.
13. T.T. Nguyen, V.A. Dahl, and J.A. Brentzen. **Cache-mesh, a Dynamics Data Structure for Performance Optimization**. Procedia Engineering, 203, 2017, 193-205.

https://doi.org/10.1016/j.proeng.2017.09.807

14. H. Valenzuela-Garcia, J.L. Molina, M.J. Lubbers, A. Garcia-Macias, J. Pampalona and J. Lerner. **On Heterogeneous and Homogeneous Networks in a Multilayered Reality: Clashing Interests in the Ethnic Enclave**, Societies 2014, 4, 85-104.
15. H.A. Reijers, I. Vanderfeesten and W.M.P. van der Aalst. **The effectiveness of workflow management systems: A longitudinal study**. International Journal of Information Management. 36(1), 2016, 126-141.
16. R.A. Aditya, D.P. Gushman, R.M. Fatchur, A.N. Freedrikson, B.P. Ari and Y. Ruldeviyani. **Master Data Management Maturity Assessment: A Case Study of Pasar Rebo Public Hospital**. International Journal of Emerging Trends in Engineering Research (IJETER). Vol. 7, Issue 5, 15-20. 2019
17. K. Verbert, N. Manouselis, H. Drachsler and E. Duval. **Dataset-Driven Research to Support Learning and Knowledge Analytics**. Educational Technology & Society, 15 (3), 2012, 133-148.
18. A. Greinera and F. Angulod. **Addressing contact tracing challenges - critical to halting Ebola virus disease transmission**. Introductory Journal of Infectious Diseases, Vol 41, 2015, 53-55. https://doi.org/10.1016/j.ijid.2015.10.025
19. P. Gastanaduy and U. Parashar. **Efficient Transmission of viral gastroenteritis in Dutch households**. The Lancet Inf. Disease . Vol. 20, Issue 5, 2020, 519-520.
20. R. Holte, A. Felner, G. Sharon, N. Sturtevant and J. Chen. **MM: A Bidirectional Algorithm that is guaranteed to meet the middle**. Artificial Intelligence. Nov. 2017, Vol. 252, 232-266
21. K. J. Karvanen, J. Vartiainen Juha, A. Timofeev and P. Jukka. **Experimental Designs for Binary Data in Switching Measurements on Superconducting Josephson Junctions**. Journal of the Royal Statistical Society Series C. Vol.56, 2007, 167-181.
22. S. S. Ganapati. **Practical Low Overhead enforcement of memory Safety for C Programs**. A PhD Dissertation in Computer and Information Sciences, University of Pennsylvania, 2012.
23. R. Kaur and N. Sharma. **A Node Authentication Mechanism to Enhance the Security in VANETs**. International Journal of Emerging Trends in Engineering Research (IJETER). Vol. 1, Issue 2, 16-22, July 2015
24. A. Amir, G. Jeriko and M. Jusmawati. **A method for calculating the sum of diagonal and anti-diagonal of power matrices without explicitly calculating its matrices**. Journal of Physics: Conference Series. 1277, 2019, 1-9 https://doi.org/10.1088/1742-6596/1277/1/012038
25. B. Barnes, E. Harris, C. Abaitey and A. Amponsah. **Construction of Symmetric Matrices using the Odd and Even Matrices**. Asian Journal of Mathematics and Computer Research. Vol. 25(4), 2018, 219-225.
26. R. Trichet, and B. Francois. **Dataset Optimization for Real-Time Pedestrian Detection**. IEEE Access 2017, 1-8
27. P. Turney and P. Pantel. **From Frequency to Meaning: Vector Space Frequency Models of Semantics**. Journal of Artificial Intelligence Res. 37, 2010, 141-188.
28. J. Hernandez, D. Garcia and F. Rabbani. **Tree Traversal to achieve e Generalization for Data Identification**. International Journal of Open Information Technologies. 2018. Vol 6, No. 11, pp7-15
29. G. Saida, and E. El-Horbaty. **An Optimization Methodology for Container Handling using Genetic Algorithm**. Procedia Computer Science, 2015. Vol 65. 662-671.
30. A. Narayan, V. Bharti and M.L Garg. **An Algorithm based on Heap of Binary Search Tree to solve Graph Coloring Problem**. International Journal of Recent Technology and Engineering. 2019. Vol 8, issue 2. 3920-3924. https://doi.org/10.35940/ijrte.B1793.078219
31. M. Tsvetovat and A. Kouznetsov. **Social Network Analysis for Startups**, O'Reilly Publishers, 2011, Sebastopol.
32. M. Zuhair and S. Kadry. **Python for Graph and Network Analysis**. Springer International Publishing. Cham, Switzerland 2017.
33. I. Cetin. **Students' Understanding of Loops and Nested Loops in Computer Programming: An APOS Theory Perspective**. Canadian Journal of Science, Mathematics and Technology Education. 2015. Vol 15, Issue 2, 155-170.
34. T. Le and C. Lin. **Bin-Picking for Planar Objects Based on Deep Learning Networks: A Case Study of USB Packs**. Sensors. 2019. 19(16). 3602.
35. O. Fidelia and O. Ogochukwu. **A framework for unified vehicle clearance and registration**. COOU Journal.of Physical Sciences, 2019, 2(8). 7-12
36. K. Nomura and M. Kainosho. **Graphical Analysis of the Relative Orientation of Molecular Alignment Tensors for a Protein Dissolved in Two Different Anisotropic Media**. Journal of Magnetic Resonance. 2002, Vol 154, No. 1, 146-153. https://doi.org/10.1006/jmre.2001.2470
37. B. Bonet and H. Geffner. **Planning as Heuristic Search.** Artificial Intelligence, 2001, 129 (1), 5-33 https://doi.org/10.1016/S0004-3702(01)00108-4