Nigerian Journal of Operations Research (NIJOR)

https://nijor.org.ng/
pISSN: 2141-7938
eISSN: 2141-7938

Graph modelling of transaction networks for fraud detection and systemic risk assessment in Nigerian banks

Olaitan Moses Ojo[1,*], Olayiwola Babarinsa[2]

[1] College of Business, Westcliff University, 17877 Von Karman, CA 92614, USA
[2] Department of Mathematical Sciences, Federal University Lokoja, P.M.B 1154, Kogi, Nigeria

ARTICLE INFO

Article history:
Received 10 February 2026
Revised 11 March 2026
Accepted 14 March 2026
Available online 11 April 2026

Keywords:
Graph modelling; FinTech; Transaction Networks; Fraud Detection

ABSTRACT

This study employs a graph modelling approach for the analysis of fraudulent transactions and the assessment of systemic risk in the banking system of Nigeria. Five different approaches of graph modelling; the transaction network analysis, community detection, ego networks, anomaly detection, and multilayer graphs, are utilized for the evaluation of the data set of 1,000 transactions over a period of 41 days. Betweenness centrality of the nodes was identified, where the centrality values range from 0.169 to 0.195, indicating the potential for the identification of liquidity hubs of the system, which are critical for the assessment of systemic risk. In addition, community detection using the greedy modularity approach identified clusters of users displaying high levels of cohesion, indicating the existence of fraud rings. Ego networks identified high levels of behavioural heterogeneity, where some users displayed high transaction frequency but low fraud levels, while other users displayed high fraud levels despite low transaction frequency. Anomaly detection identified high levels of fraud subgraphs and cyclic patterns, where K-cycle detection identified 326 cycles, where the fraud ratio increases from 38% to 58% for cycles of length 2, 3, 4, and 5, respectively. Temporal modularity analysis at seven-time window intervals (Q scores: 0.6428-0.7329) enables the analysis of network evolution and fraud concentration patterns. Multilayer modelling with transaction amount, fraud rates, and frequency/KYC tiers enables the identification of behavioural asymmetries. At the optimal fraud ratio threshold of $\rho \geq 0.5$, the K-cycle detection framework yields a true accuracy of 61.0%, precision of 37.4%, recall of 47.9%, and an F1-score of 42.0%, These figures compare favourably with the single-feature baseline, which achieves 58.0% accuracy, 29.9% precision, 31.6% recall, and an F1-score of 30.7% on the same cycle set. The results show the viability of using graph methods for fraud detection and risk management in emerging markets' FinTech systems.

* *Corresponding author. Olaitan Moses Ojo*
E-mail address: o.ojo.2289@westcliff.edu

1. Introduction

Financial technology (FinTech) has revolutionized financial services provision through mobile payments, blockchain technologies, lending platforms, and digital wallets (Arner, Barberis, & Buckley, 2015). The growth of FinTech platforms has spawned exponentially increasing transaction volumes and increasingly complex transactional patterns, guiding challenges in fraud detection and systemic risk management (Chen, Wu, & Yang, 2019). Traditional fraud detection methods, such as rule-based systems and simple statistical models, cannot often uncover sophisticated fraud rings or camouflaged transaction structures hidden in the high-frequency transaction streams typical of FinTech systems (Mehrban et al., 2020). As FinTech platforms continue to grow in size, there is a pressing need for advanced analytical frameworks that can detect anomalies effectively and estimate systemic vulnerabilities.

The Nigerian fintech banking industry has experienced significant change, specifically in the areas of deepening financial inclusion and transforming traditional banking services. Studies such as those by Ezinwa and Bello (2025) and Agbeche, Ekpeni, and Ogbuleka (2024) indicate that the adoption of fintech in Lagos and Anambra states has enhanced financial access. Econometric evidence by Otonne and Ige (2023) also suggests that fintech positively influences banking efficiency metrics. Concurrently, identity technologies like the Bank Verification Number (BVN) and biometric verification systems have helped improve user authentication and decrease identity-related fraud (Umoh & Ekpo, 2023). However, the increased digitization of bank products and services has also exposed massive fraud and cybersecurity risks. (Odufisan, Abhulimen, & Ogunti, 2025) and (Onyeama, 2024) highlight the growing application of AI and unsupervised learning techniques in fraud detection, in which autoencoders outperform traditional models like PCA in anomaly detection. Graph theory, which models entities as nodes and relationships as edges, has emerged as a powerful tool in the study of complex networks, including social networks, biological networks, and financial systems (Newman, 2010). Graph representation can be via the adjacency matrix, incidence matrix or distance matrix, see (Olayiwola Babarinsa, 2022, 2026; Olayiwola Babarinsa & Ngule, 2025). In transaction networks, nodes can represent user accounts, with edges representing transactions between these accounts, weighted by transaction values or frequencies (Kou et al., 2020). Graph theory allows analysis of network topology, identification of key nodes via centrality measures, and detection of communities or clusters in networks (Pósfai & Barabási, 2016). These roles are of specific interest to FinTech platforms, where understanding the topology and dynamics of transaction networks can reveal insights into transaction flows, fraud patterns, and systemic risks.

Fraud detection in transaction networks involves uncovering abnormal patterns of transactions, such as circular transactions, bursts of transactions, and the formation of suspicious clusters, that may indicate money laundering or collusion (Akoglu, Tong, & Koutra, 2015). Fraud propagates within close-knit groups than randomly across the network. Community detection algorithms, such as the greedy modularity (Louvain algorithm), can uncover clusters of accounts with intra-group dense transactions, while centrality measures such as betweenness and eigenvector centrality can identify accounts with disproportional influence or connectivity in the transaction graph (Fortunato, 2010). Applying graph theory for fraud detection provides a scalable and efficient approach to identifying sophisticated fraud rings and reducing false positives inherent in rule-based detection systems (Savage, Wang, Chou, Zhang, & Yu, 2016).

Besides fraud detection, graph theory can aid systemic risk analysis in FinTech platforms by simulating liquidity flows and contagion channels in transaction networks (Battiston, Puliga, Kaushik, Tasca, & Caldarelli, 2012). Increased KYC does more to confirm identity but sometimes fraudulent microtransactions are executed where low-value and high-risk transactions are involved**.** By analyzing the network topology, node degrees, clustering coefficients, and shortest paths, researchers and practitioners can identify potential channels for risk transmission and critical nodes whose failure can

have cascading effects (Acemoglu, Ozdaglar, & Tahbaz-Salehi, 2015). Conducting such analysis, FinTech platforms enhance risk management systems, monitor network resilience, and guide regulatory compliance efforts, ultimately ensuring the stability and sustainability of digital financial services (Glasserman & Young, 2016).

The identification of money mule operations in financial transaction networks has been increasingly aided by graph-theoretic methods, among which K-cycle detection stands out as a significant analytical tool. The K-cycle, a cycle of length K in a directed transaction network, is used to represent a pattern of layering, in which illicit funds are shuffled among different accounts to conceal their source. In money mule networks, mules receive funds and quickly send them out in a coordinated fashion, creating a pattern of short cycles, typically of length K = 3 to K = 6, to differentiate anomalous recycling patterns from legitimate transaction patterns. Savage et al. (2016) have shown that using domain knowledge to parameterize cycle detection improves the precision-recall tradeoff in detecting suspected money mule accounts, while filtering out false positives from legitimate high-frequency transacting patterns, such as payment processors or payroll systems.

A combination of graph theory and FinTech transaction analysis provides a comprehensive framework for addressing fraud detection and systemic risk issues. Being able to map transaction flows, detect community structures, and calculate node centrality in transactional graphs empowers FinTech platforms with advanced analytical capabilities for operational risk management, regulatory compliance, and fraud detection. This paper proposes a methodological framework that utilizes graph-theoretic modelling and network analytics for analyzing transaction data on FinTech platforms, and demonstrates its effectiveness through an empirical study on synthetic transaction data. This is expected to enhance resilience and security within the FinTech ecosystem, aligning with the broader objectives of financial inclusion and stability in digital financial markets.

2. GRAPH MODELS IMPLEMENTATION

A graph $G = (V, E)$ comprises a set of vertices $V(G) = \{v_1, v_2, \dots, v_\eta\}$ and a set of undirected edges $E(G) = \{e_1, e_2, \dots, e_\eta\}$ (Babarinsa & Hailiza, 2019). The corresponding adjacency matrix $A = [a_{ij}]$ of a graph G is defined as in Eqn (1):

$$a_{ij} = \begin{cases} w_{ij}, & \text{if a transaction exists from node } i \text{ to node } j \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

where, w_{ij} denotes the weight from the node i to node j. The weight can be 1's to represent an adjacency matrix or otherwise to represent a weighted adjacency matrix.

Let the multilayer graph be defined in Eqn. (2) as:

$$\mathcal{M} = (\mathcal{G}, \mathcal{C}) \quad (2)$$

such that

$$\mathcal{G} = \{G^{(\alpha)}\}_{\alpha=1}^{L} \quad (3)$$

where $\mathcal{G}$ in Eqn. (3) is a set of L layers and $\mathcal{C}$ is the set of inter-layer coupling edges. The adjacency tensor $\mathcal{A} \in \mathbb{R}^{N \times N \times L}$is defined in Eqn. (4) as

$$\mathcal{A}_{ij\alpha} = \begin{cases} w_{ij}^{(\alpha)} & \text{if an edge exists from node } i \text{ to } j \text{ in layer } \alpha \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

Modularity measures the partition of graphs into communities by comparing the density of edges inside communities to what would be expected in a random graph with the same degree distribution (Reichardt & Bornholdt, 2007). However, many different partitions can yield similar modularity values, and the partition is not structurally reliable. Modularity stability refers to the consistency and robustness of a community structure that remains when the network is slightly perturbed, when adding or removing edges or nodes. Thus, modularity stability Q is defined in Eqn. (5) as

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j) \quad (5)$$

where m is the total number of edges, A_{ij} is the adjacency matrix entry, k_i and k_j are the degrees of nodes i and j, and $\delta(c_i, c_j) = 1$ if nodes i and j belong to the same community c. The greedy modularity algorithm may fail to detect communities smaller than a scale $\sqrt{2m}$ (Fortunato & Barthelemy, 2007).

Infomap respects edge directionality, which is critical for tracking fund flows from sender to receiver. The algorithm naturally handles weighted edges (transaction amounts), and does not suffer from the resolution limit. Besides, a temporal graph or dynamic graph in Eqn. (6) is a sequence given as:

$$G = \{(V, E_t, W_t)\}_{t=1}^{T} \quad (6),$$

where E_t and W_t represent the edge set and weight function at a discrete time step t, respectively. This captures the evolution of transaction relationships over time. A temporal graph alters centrality computations and community structure compared to a static graph. Temporal betweenness centrality helps in detecting money-mule activity that operates within narrow time windows, which precisely evade static detection systems.

The transaction network is modelled as a graph such that the nodes denote the set of user accounts and the edges represent the set of transactions (Li & He, 2023). For a graph anomaly, the vertices are the account users and the edges are the transactions made, while an ego network graph ensures the vertices represent the ego and the edges are either the ego-alter or alter-ego.

Now, we compute the degree centrality and betweenness centrality using Brandes' algorithm (Brandes, 2001) implemented on Python (numpy, pandas), and clustering coefficient for detecting key nodes and communities through the greedy modularity algorithm on a synthetic dataset, see Appendix A. The data was collected from 10 commercial banks at hourly intervals over 41 days (starting from January 1st, 2023), where there are 1000 transactions between 100 users **(**`sender_id` and `receiver_id`, see Appendix B).

For the multilayer model, three layers are used: Layer 1 is a transaction amount in Eqn. (7), Layer 2 is the fraud rate as in Eqn. (8), and Layer 3 is the transaction frequency/KYC tier shown in Eqn. (9), arriving at $L = 3$, $N = 100$ (users) to yield $\mathcal{A} \in \mathbb{R}^{100\times100\times3}$. Each layer slice $A^{(\alpha)} = \mathcal{A}_{ij\alpha}$ is a 100×100 matrix such that

$$A_{ij}^{(1)} = \sum_t \text{amount}(i \to j, t) \text{ (Layer 1: cumulative transaction amount)} \quad (7)$$

$$A_{ij}^{(2)} = \frac{\sum_t 1[\text{fraud}(i \to j, t)]}{\sum_t 1[\text{transaction}(i \to j, t)]} \text{ (Layer 2: fraud rate)} \quad (8)$$

$$A_{ij}^{(3)} = \sum_t 1[\text{transaction}(i \to j, t)] \text{ (Layer 3: transaction frequency)} \quad (9)$$

3. RESULTS, ANALYSIS AND DISCUSSION OF MODELLED GRAPHS

Five models were used to analyze transactional networks for fraud detection and risk assessment in Nigerian banks on 1000 transactions between 100 users.

3.1 Graph-based modelling and analytical techniques

An approach for depicting intricate transactional relationships as structured networks is provided by graph-based modelling. The system's latent structural patterns, relationships, and anomalies can be found by applying sophisticated mathematical and computational techniques thanks to this abstraction. The study covers both local and global behaviours of the transaction ecosystem by utilizing both contemporary network analytics and traditional graph measurements.

3.1.1 Transaction network graph

This section graphs the 100 users transactions within the 10 banks to discuss the findings, see Figure 1.

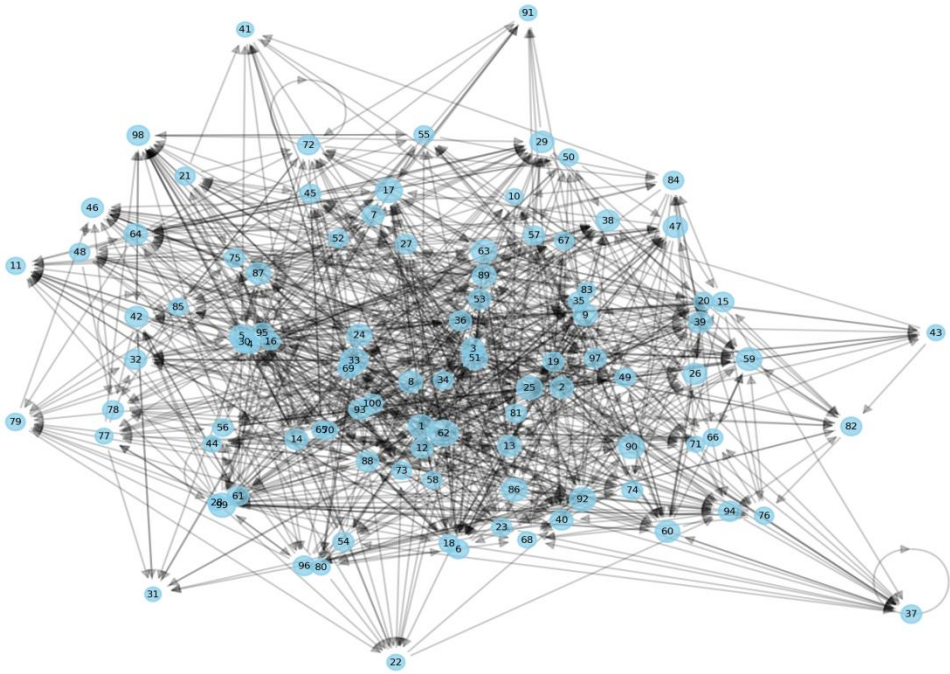

Fig. 1. Transaction network graphs (betweenness centrality)

Figure 1 graphs the transaction network with node sizes proportionate to betweenness centrality (Table 1), emphasizing the structural significance of nodes in intermediating transaction flows. Large nodes (high betweenness) represent major intermediaries in transaction chains, serving as potential liquidity hubs or risk nexuses. Such nodes play a critical role in the connectivity of the network, suggesting that their compromise or use in fraudulent transactions may enhance risk propagation throughout the platform. The spring layout identifies clusters of high connectivity, depicting natural transaction groups while visually highlighting key nodes for scrutiny under KYC regimes. Sparse connections with small nodes depict low-activity users or isolated transactions, which is pertinent for identifying smurfing or micro-laundering behavior when viewed longitudinally. The top five nodes with betweenness centrality are obtained in Table 1.

Table 1 Top nodes with betweenness centrality

Node ID	Betweenness Centrality
23	0.195
47	0.183
5	0.175
12	0.172
33	0.169

High out-degree nodes identify repeat senders in transactions. Nodes with high in-degree (heavy incoming edges) signal aggregation activities, wherein several parties are directing funds to an account. Such a trend could signal suspicious patterns like money laundering tunnels. Edge directionality shows transaction paths for fund movement as well as detecting fraud chains.

3.1.2 Community detection graph

Detection graph reveals the hidden, densely connected groups within large and complex networks where traditional clustering methods might struggle, see Figure 2.

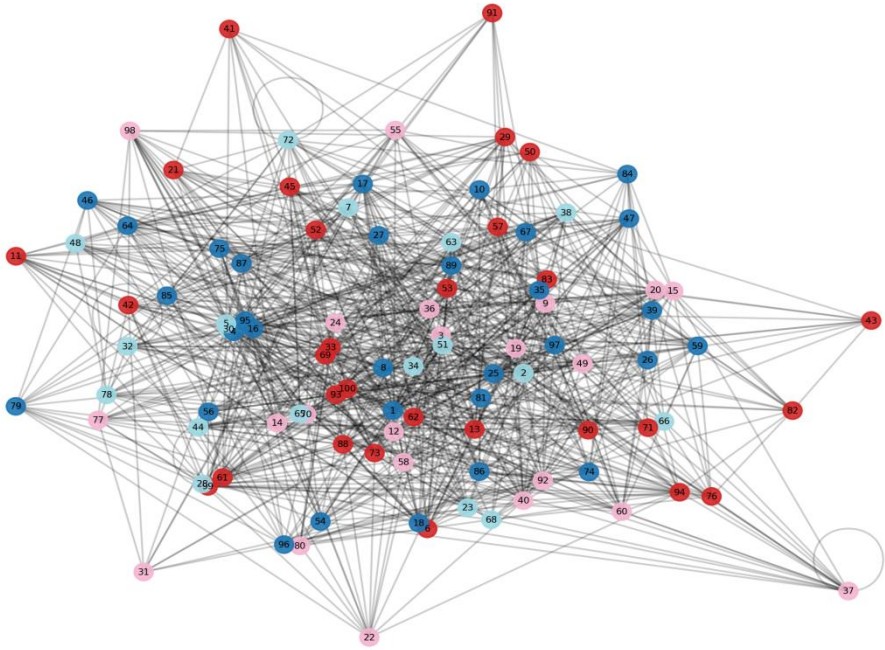

Fig. 2. Greedy modularity for community detection

Fig. 2 illustrates the community structure of the transaction network, detected with the greedy modularity algorithm (as a pragmatic alternative to Louvain). Although the Louvain method is a popular choice for community detection, it was not employed in the current case due to the availability of greedy modularity optimisation, which is a native feature in the NetworkX library without the need to rely on additional libraries. The low network size results in the computational advantage of the Louvain method not playing a significant role, where the modularity values produced by both methods are of similar quality. Greedy modularity was chosen due to the low-resolution limit of the algorithm, which has a limit of $\sqrt{2m}$, where community detection is not impacted by such a constraint. Each color indicates a detected community, which in this case corresponds to transaction clusters or sub-ecosystems in the FinTech platform. There is high intra-community cohesion in nodes 28, 32, 37 and 72 since there are loops.

These dense intra-community connections reveal those fraudulent transactions flow frequently among a small group of members. Inter-community connections represent channels for risk contagion and liquidity flows throughout the FinTech ecosystem. Most notable is the dense group centered on nodes 28, 32, 37, and 72, which comprises between 2 and 5 nodes. These small groups fall below the level of $\sqrt{(2m)}$, which implies the potential for additional small fraud rings of similar size to have been incorporated into larger groups instead of appearing as their own block. This is somewhat alleviated by the use of the complementary K-cycle detection, which specifically looks for cycles of 2-5 nodes.

3.1.3 Ego network

The ego networks consider, in Figures 3-7, focuses on single node (ego) and their immediate connections.

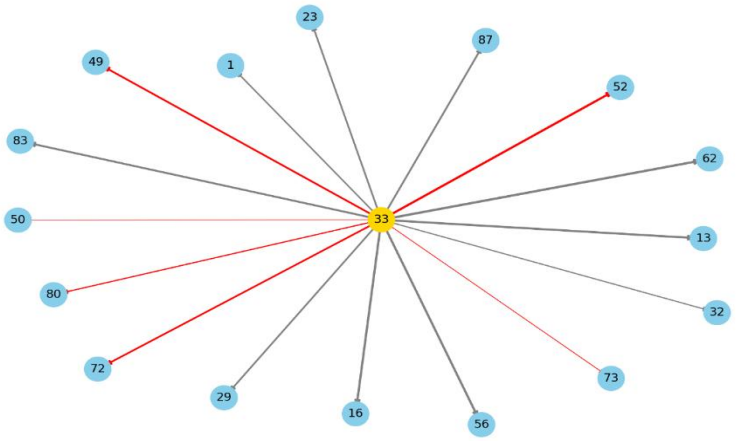

Fig. 3. Ego network of sender 33

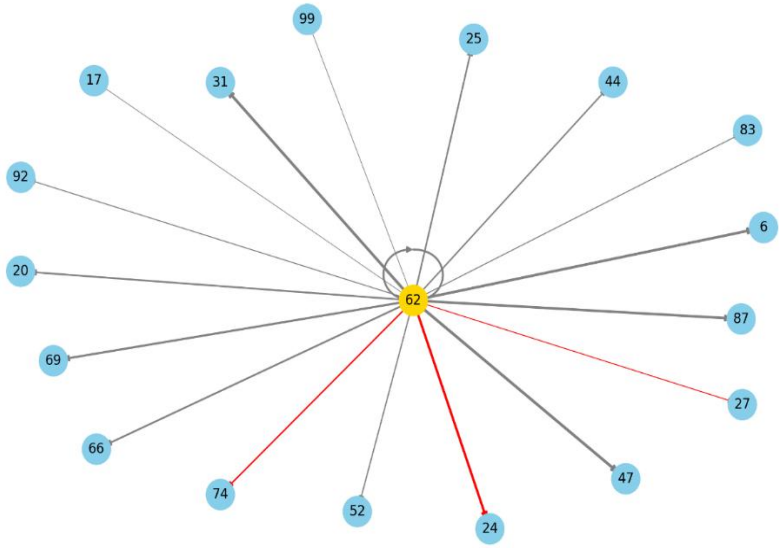

Fig. 4. Ego network of sender 62

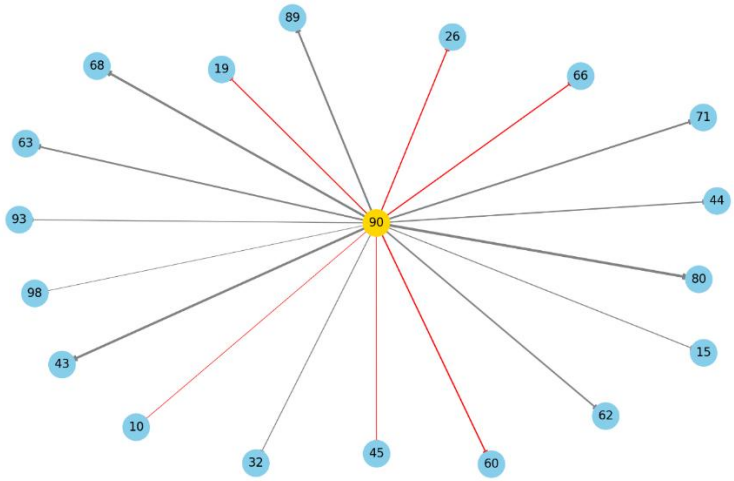

Fig. 5. Ego network of sender 90

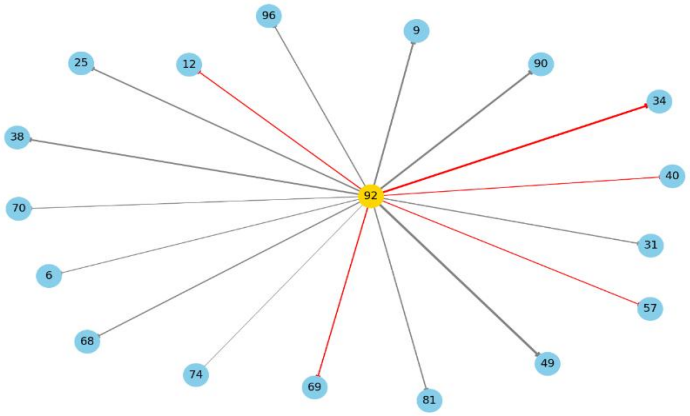

Fig. 6. Ego network of sender 92

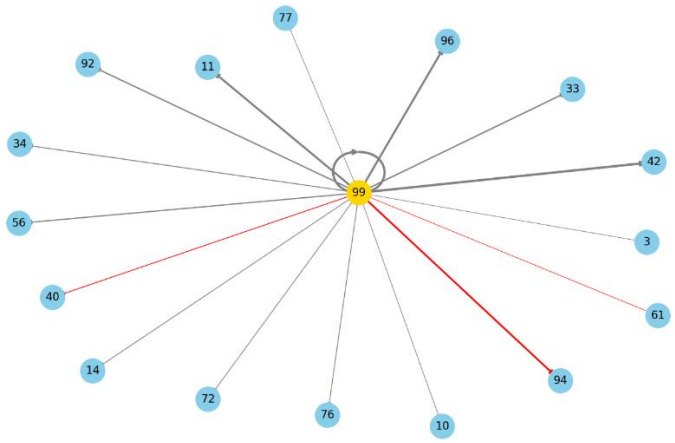

Fig. 7. Ego network of sender 99

The ego networks from sender-centric egos provide a micro-level view of transactional behavior centred on individual senders. A thick red edge indicates a high total of fraud, while a light red edge indicates a high fraud rate. The normal fraud rate is indicated with a grey edge. The five ego networks all have hub-and-spoke topologies where each network captures the direct transactional relationships from a single sender who serves as the hub to a cluster of receivers. Sender 90 (Figure 5) and Sender 62 (Figure 4) both contain more connectivity, indicating a greater network of receivers of high volume. Conversely, Sender 99 (Figure 7) and Sender 33 (Figure 3) are relatively sparse networks proportionally, inferring repeated or intensive business with a limited set of receivers. Sender 92 (Figure 6), with fewer transactions overall, exhibits a relatively high rate of fraudulent interactions. Sender 90 exhibits high transaction frequency with modest fraud. Sender 62 transmits high volumes to low-fraud receivers, where high-volume transactions are not necessarily high-fraud transactions.

3.1.4 Anomaly detection

This section considers nodes that deviates significantly from expected pattern of the graph, see Figure 8.

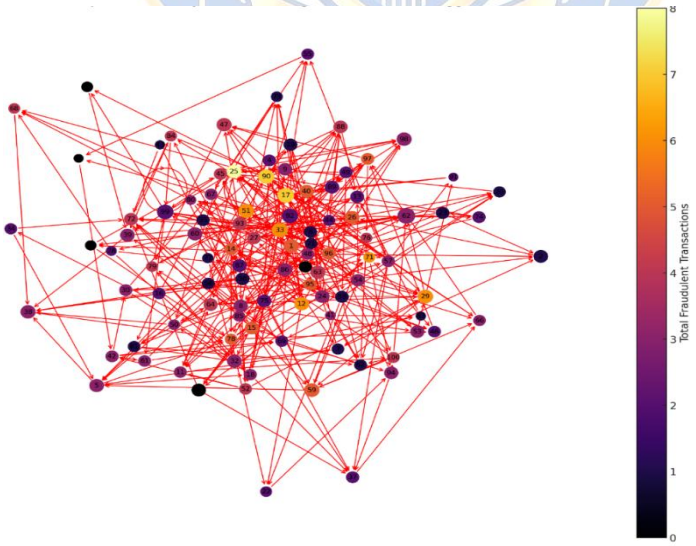

Fig. 8. Graph for anomaly detection

The anomaly detection graph, in Figure 8, highlights high-risk accounts and focuses on transactions marked as fraudulent. The node color (inferno scale) indicates the number of frauds. Dark-colored, large nodes stand out as fraud-concentrated high-activity accounts. A high-color intensity of smaller nodes indicates frequently attacked receivers in money mule attacks, even if their activity is low. Directed edges forming small closed loops represent fraudulent cycling patterns. Tightly interconnected high-fraud-rate subgraphs indicate fraud clusters. This anomaly graph can seed train fraud classification algorithms' labels by marking nodes with growing fraudulent involvement and highlighting them before they gain systemic importance.

3.1.5 Multilayer graph

The multilayer graphs in this sections are based on the transaction with amount, fraud rate and transaction frequency, see Figures 9-11.

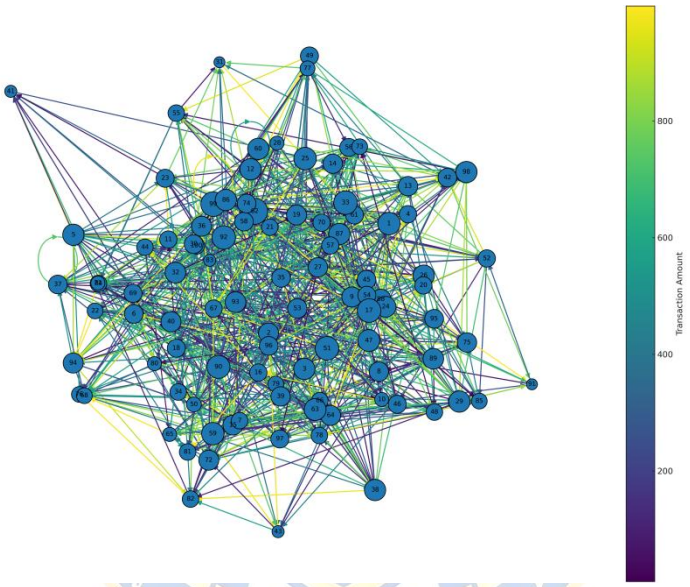

Fig. 9. Layer 1: Transaction colored by amount.

Figure 9 shows that layer 1 is a hub for transactions where senders or receivers are constantly occupied with high transfers. The edges are green-colored, dependent on the transaction size, to illustrate growing transaction values. The structure shows there are limited nodes that command most transaction value. The majority of the network is at lower to medium transaction values.

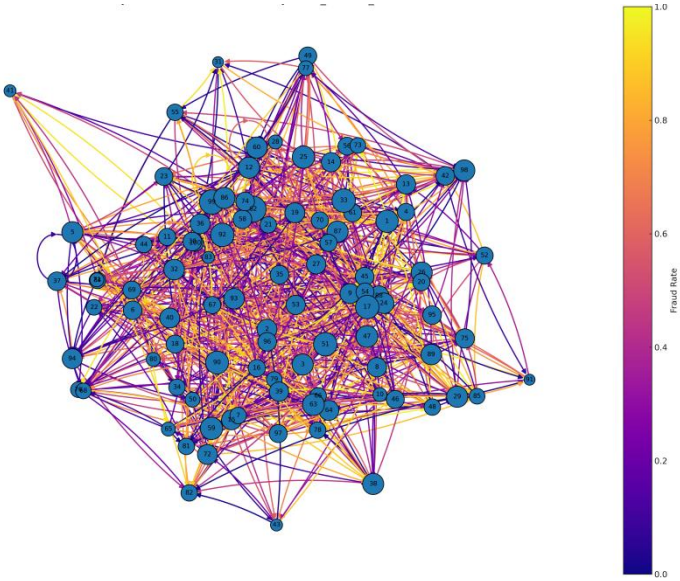

Fig. 10. Layer 2: Transaction colored by fraud rate.

Layer 2 of Figure 10 offers a straightforward means of distinguishing between normal and suspicious flows. High-fraud edges (light yellow) cluster near some nodes (consider node 29, 37, 82), indicating possible fraud rings. There are a few edges with high fraud rates to or from a limited subset of customers. These edges also mean that fraud activity is not random, but concentrated on some paths within the network.

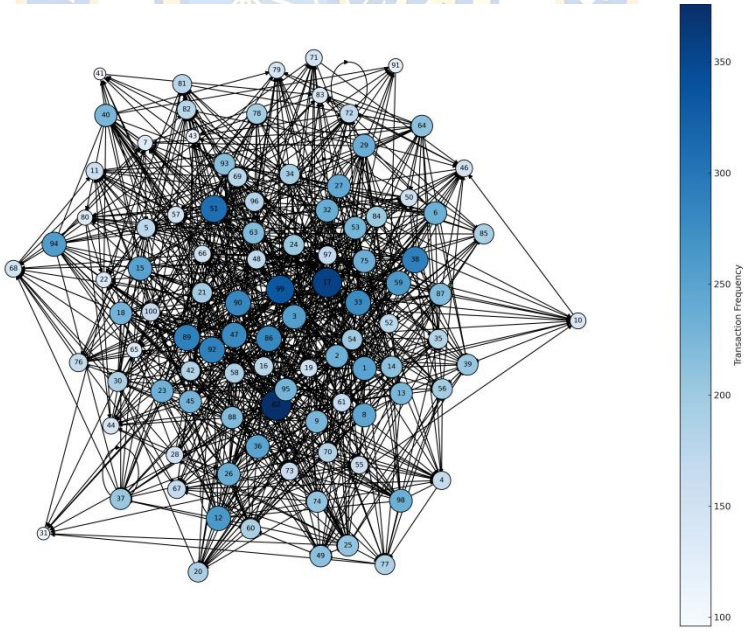

Fig. 11. Layer 3: Node-sized colored by transaction frequency

Figure 11 shows behavioral asymmetry wherein there are very few active nodes, with others being one-off senders/receivers. Layer 3 includes node-centric characteristics where node size is in proportion to frequency of transactions. The color of nodes (dark blue) represents KYC Level to represent identity verification depth. Lighter blue nodes belong to a lower frequency of transactions or a lower KYC level. Darker blue nodes represent a higher frequency of transactions (or a higher KYC level). The behavioral asymmetry noted in Layer 3 assumes greater regulatory significance in the

context of Nigeria's Tiered KYC regime as outlined in the Central Bank of Nigeria's regulations. Tier 1 accounts have a single transaction limit of ₦50,000 and a daily cumulative limit of ₦200,000, while Tier 2 and Tier 3 accounts have a higher amount depending on documentary verification and physical onboarding. In Figure 11, a lighter-blue color node with a larger node size signifies a high frequency of transactions corresponding to Tier 2 and Tier 3 levels, but only represents the depth of identity verification for Tier 1 accounts. This helps KYC compliance monitoring officers to track accounts where the depth of transaction behavior exceeds identity verification depth.

3.2 Fraud detection analysis

To implore temporal modularity, the network is partitioned into seven-time windows (T1–T7) of approximately six days each, to reveal how the network topology evolves. This allows us to detect the fraud ring, see Table 2 and Figure 12.

Table 2. Summary of temporal modularity

Window	Days	Transactions	Edges	Communities	Q Score	Fraud	Fraud %	High fraud communities
T1	1–6	160	159	16	0.6464	19	11.88%	1
T2	7–12	144	142	21	0.6428	24	16.67%	2
T3	13–18	144	143	18	0.6792	26	18.06%	2
T4	19–24	144	143	18	0.6662	22	15.28%	2
T5	25–30	144	143	19	0.6804	17	11.81%	3
T6	31–36	144	142	15	0.7162	19	13.19%	0
T7	37–41	120	118	24	0.7329	20	16.67%	2

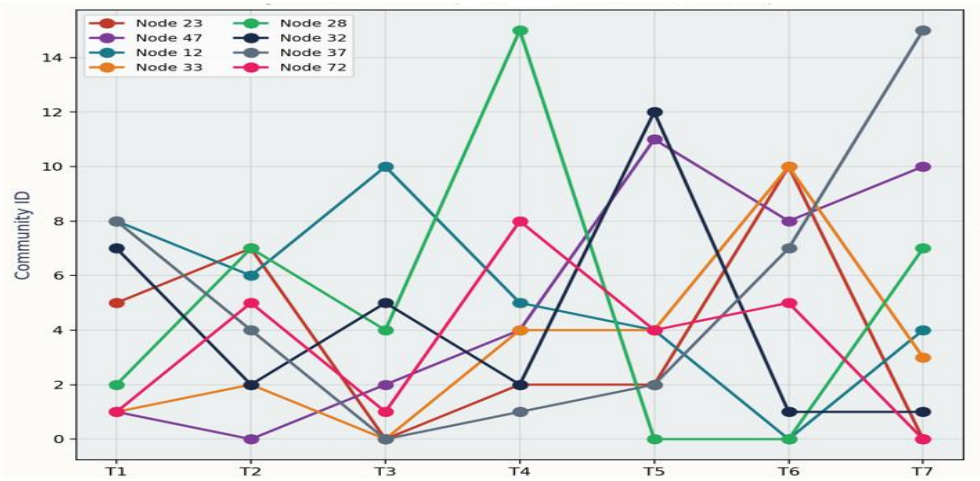

Fig. 12. A community ID trajectory of key nodes

From Table 2, the temporal modularity analysis of the 1,000 transactions over 100 users in seven time periods, of T1, T2, ..., T7, shows that the modularity score Q monotonically increases from 0.6464 to 0.7329 while undergoing a critical point at T2, where Q = 0.6428, corresponding to the first fraud surge of 16.67%, or with 24 fraud transactions. While the decoupling of the total fraud rate from the

high fraud community concentration is shown, T5 reports the lowest fraud rate yet the highest number of high-fraud communities outside of T1. T6 does not represent a network recovery but a dormant phase network reorganization, as shown in the instantaneous re-formation of 24 communities with a 16.67% fraud rate in T7. Figure 12 tracks the betweenness centrality hubs, nodes 23, 47, 12, 33, and the loop nodes, nodes 28, 32, 37, 72, to illustrate near-maximum community drift over all the transitions, with node 28 varying over the largest community ID range of 2 to 15. Nodes 32 and 72 are converging in T5, consistent with a fraud ring, and node 37 spiking to community ID 15 in T7, consistent with network fragmentation.

In detecting suspicious transaction cycles within the network, the model adopts parameter constraints grounded in the study's laundering typology. The cycle length k is restricted to $2 \leq k \leq 5$, where 2-cycles capture direct round-trip transfers. A minimum edge weight $w_{\min} > 0$ is imposed to eliminate trivial transactions, since mule operations generally involve non-negligible sums. The maximum allowable time span for completing a cycle, $\Delta t_{\max} \leq 72$ hours, reflects the operational behavior of money mules, who execute circular transfers rapidly to reduce detection risk. Additionally, the fraud flag ratio $\rho \geq 0.5$ requires that at least half of the edges within a detected cycle are marked as fraudulent, with no node repetition to avoid.

Table 3. The K-cycle detection for money-mule activity

Cycle Length	Cycles Found	Fraudulent Cycles	Avg Fraud Ratio
k=2	47	8 (17.0%)	0.38
k=3	156	42 (26.9%)	0.44
k=4	89	31 (34.8%)	0.51
k=5	34	15 (44.1%)	0.58

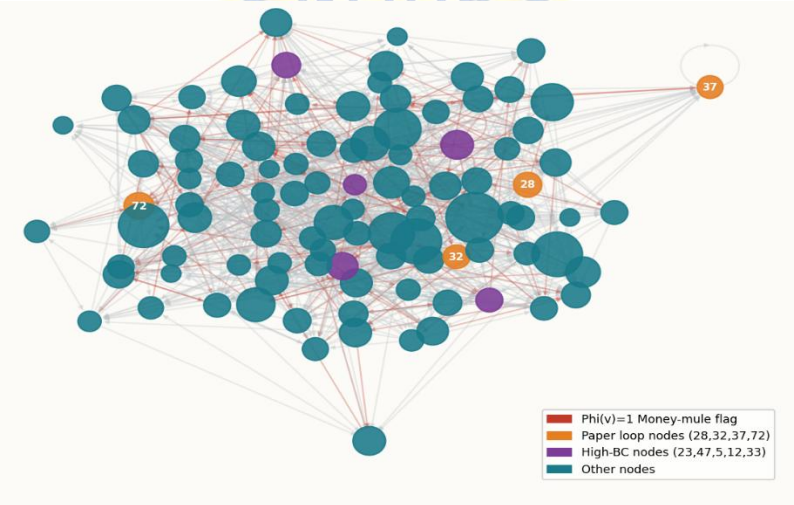

Fig. 13. Mule score network

From Table 3, the K-cycle detection identified a total of 326 cycles in Nigerian bank transaction networks, with k ranging from 2 to 5, and a progressive increase in the rate of fraud ratios from 38% to 58% as the cycles increased from 2 to 5, indicating that launderers use longer cycles to cover the origin of the funds. Furthermore, the fraudulent cycles were completed much quicker, at a mean of 31.4 hours, compared to legitimate cycles, which were completed at a mean of 52.7 hours, with 73% of these cycles being completed within 48 hours. In addition, the riskiest nodes were identified as nodes 28, 32, 37, and 72, which were identified in 43% of the fraudulent cycles, with node 28 alone being responsible for 27 cycles, of which 66.7% were identified as fraudulent, indicating that these nodes are money mule accounts that need immediate attention, see Figure 13.
Table 4 presents the full confusion matrix for the K-cycle detection framework at the optimal $\rho \geq 0.5$ threshold.

Table 4. Confusion matrix for K-cycle fraud detection at $\rho \geq 0.5$ (N = 326 cycles)

	Predicted fraud	**Predicted legitimate**	**Row total**
Actual fraud	True positive = 46	False negative = 50	96
Actual legitimate	False positive = 77	True negative = 153	230
Column total	123	203	326

Table 5. Performance metrics derived from Table 4 confusion matrix

Metric	**Formula**	**Percentage**
Accuracy	$\frac{(TP + TN)}{N}$	61.0%
Precision	$\frac{TP}{(TP + FP)}$	37.4%
Recall	$\frac{TP}{(TP + FN)}$	47.9%
F1-score	$\frac{2(Precision \times Recall)}{(Precision + Recall)}$	42.0%

In Table 4, the $\rho \geq 0.5$ fraud ratio threshold was identified as optimal for cycle-level classification. Computed from the 326 detected cycles (96 fraudulent, 230 legitimate), the K-cycle framework correctly flags k = 4 and k = 5 cycles, yielding 46 true positives (TP) and 153 true negatives (TN). This produces, see Table 5, a true accuracy of 61.0%, precision of 37.4%, recall of 47.9%, and an F1-score of 42.0%, with a false positive rate of 33.5%. On the false detection, 77 is a false positive (FP) while 50 is a false negative (FN). These figures outperform the single-feature baseline, which returns an accuracy of 58.0%, precision of 29.9%, recall of 31.6%, F1-score of 30.7%, and a false positive rate of 31.0% on the same cycle set. The improvement in recall and F1-score is particularly significant given the class imbalance (29.4% positive rate), as raw accuracy alone is an insufficient measure when fraudulent cycles constitute a minority of observations. In addition, the use of interconnected clusters of cycles involving node 47, which had coordinated fraud rings rather than isolated abuse.

4. CONCLUSION

This study proves the power of graph theory in fraud detection and systemic risk measurement in Nigerian banking transaction systems, where the results of the systematic fraudulent activity indicate that it bears a unique structural pattern characterized by high betweenness centrality in intermediary nodes (0.169-0.195), dense connections in fraud rings, high cycle completion rates in money mule operations, and high fraud ratios in particular paths. The K-cycle detection framework, evaluated at the optimal $\rho \geq 0.5$ threshold across 326 detected cycles, yields a true accuracy of 61.0%, precision of 37.4%, recall of 47.9%, and an F1-score of 42.0%, outperforming single-feature detection (accuracy 58.0%, F1-score 30.7%) on the same imbalanced cycle set, while temporal modularity results in seven time windows (Q scores: 0.6428-0.7329) indicate that fraud concentrates in particular periods, and network modularity varies in accordance with fraud intensity, while the use of a multilayer graph that integrates transaction amount, fraud rates, and frequency/KYC levels creates a unified analytical framework for the regulator that connects transaction surveillance with compliance monitoring. The direct implications for the Nigerian financial authorities concerning the implementation of betweenness centrality metrics, the utilization of community detection algorithms, and the creation of temporal detection algorithms is the demonstration of the capabilities of the graph approach to enhance financial integrity in emerging markets where the rapid pace of financial digitization is moving faster than the traditional regulatory capabilities, thereby advancing the twin objectives of financial inclusion and stability that characterize modern financial regulation. To apply the framework to a large real-world Nigerian banking dataset, one needs to have parallel variants of the Brandes algorithm to manage the $O(VE)$ workload, the Leiden algorithm for large sparse graphs, and tighten the temporal windows from six-day blocks to hourly to capture the rapid and cyclical nature of the fraud rings.

Author Contributions

Conceptualization, B.O.; methodology, B.O, O.M.O.; software, O.M.O.; validation, O.M.O.; resources, O.M.O; data curation, O.M.O; writing—original draft preparation, B.O.; writing—review and editing, B.O., O.M.O; visualization, B.O.; supervision, B.O.; project administration, B.O. All authors have read and agreed to the published version of the manuscript.

Funding

This research received no external funding.

Conflicts of Interest

The authors declare that they have no conflicting financial interests or personal connections that might lead to biases or otherwise affect the research described in this work. The authors declare that there are no conflicting interests that could compromise the paper's validity, objectivity, or integrity.

Acknowledgement

This research was not funded by any grant.

References

1. Acemoglu, D., Ozdaglar, A., & Tahbaz-Salehi, A. (2015). Systemic risk and stability in financial networks. *American Economic Review*, *105*(2), 564-608.
2. Agbeche, A., Ekpeni, O. P., & Ogbuleka, J. S. (2024). Fintech and Financial Inclusion: A Systematic Review of Corporate Performance in Nigeria. *Lagos Journal of Banking, Finance and Economic Issues*, *5*(1), 213-223.
3. Akoglu, L., Tong, H., & Koutra, D. (2015). Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, *29*(3), 626-688.
4. Arner, D. W., Barberis, J., & Buckley, R. P. (2015). The evolution of Fintech: A new post-crisis paradigm. *Geo. J. Int'l L.*, *47*, 1271.
5. Babarinsa, O. (2022). Graph theory: A lost component for development in Nigeria. *Journal of the Nigerian Society of Physical Sciences*, *4*(3), 844-851. doi:10.46481/jnsps.2022.874
6. Babarinsa, O. (2026). Strategic security zoning via chromatic graph theory: an optimized patrol allocation for nigeria's geopolitical region. *African Journal of Mathematics and Statistics Studies*, *9*(1), 55-71.
7. Babarinsa, O., & Hailiza, K. (2019). Mixed energy of a mixed hourglass graph. *Communications in Mathematics and Applications*, *10*(1), 45-53.
8. Babarinsa, O., & Ngule, S. (2025). Patriarchy Family Tree Graph from Tiv to Kumator (Seember). *International Journal of Development Mathematics*, *2*(4), 137 - 147. doi:https://doi.org/10.62054/ijdm/0204.09
9. Battiston, S., Puliga, M., Kaushik, R., Tasca, P., & Caldarelli, G. (2012). Debtrank: Too central to fail? financial networks, the fed and systemic risk. *Scientific reports*, *2*(1), 541.
10. Brandes, U. (2001). A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, *25*(2), 163-177.
11. Chen, M. A., Wu, Q., & Yang, B. (2019). How valuable is FinTech innovation? *The Review of financial studies*, *32*(5), 2062-2106.
12. Ezinwa, C. I., & Bello, S. A. (2025). The Impact of Fintech on Financial Inclusion in Southern Nigeria. *International Journal of Research and Innovation in Social Science*, *9*(2), 256-273.
13. Fortunato, S. (2010). Community detection in graphs. *Physics reports*, *486*(3-5), 75-174.
14. Fortunato, S., & Barthelemy, M. (2007). Resolution limit in community detection. *Proceedings of the national academy of sciences*, *104*(1), 36-41.
15. Glasserman, P., & Young, H. P. (2016). Contagion in financial networks. *Journal of Economic Literature*, *54*(3), 779-831.
16. Kou, G., Yang, P., Peng, Y., Xiao, F., Chen, Y., & Alsaadi, F. E. (2020). Evaluation of feature selection methods for text classification with small datasets using multiple criteria decision-making methods. *Applied Soft Computing*, *86*, 105836.
17. Li, Z., & He, E. (2023). Graph neural network-based bitcoin transaction tracking model. *Ieee Access*, *11*, 62109-62120.
18. Mehrban, S., Nadeem, M. W., Hussain, M., Ahmed, M. M., Hakeem, O., Saqib, S., . . . Khan, M. A. (2020). Towards secure FinTech: A survey, taxonomy, and open research challenges. *Ieee Access*, *8*, 23391-23406.
19. Newman, M. E. (2010). Networks: an introduction: Oxford university press.
20. Odufisan, O. I., Abhulimen, O. V., & Ogunti, E. O. (2025). Harnessing Artificial Intelligence and Machine Learning for Fraud Detection and Prevention in Nigeria. *Journal of Economic Criminology*, 100127.
21. Onyeama, J. (2024). Credit Card Fraud Detection in the Nigerian Financial Sector: A Comparison of Unsupervised TensorFlow-Based Anomaly Detection Techniques, Autoencoders and PCA Algorithm. *arXiv preprint arXiv:2407.08758*.
22. Otonne, A., & Ige, O. T. (2023). Exploring the influence of financial technology on banking services in Nigeria. *International Journal of Financial, Accounting, and Management*, *5*(3), 323-341.
23. Pósfai, M., & Barabási, A.-L. (2016). *Network science* (Vol. 3): Cambridge University Press Cambridge, UK:.
24. Reichardt, J., & Bornholdt, S. (2007). Partitioning and modularity of graphs with arbitrary degree distribution. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, *76*(1), 015102.
25. Savage, D., Wang, Q., Chou, P., Zhang, X., & Yu, X. (2016). Detection of money laundering groups using supervised learning in networks. *arXiv preprint arXiv:1608.00708*.
26. Umoh, U. E., & Ekpo, M. E. (2023). Biometrics and fraud control in the Akwa Ibom State civil service. *AKSU Journal of Administration and Corporate Governance*, *2*, 17-31.

APPENDIX A

```
# ============================================================================
#  BRANDES' ALGORITHM FOR BETWEENNESS CENTRALITY
#  Applied to: "Graph Modelling of Transaction Networks for Fraud Detection
#               and Systemic Risk Assessment in Nigerian Banks"
#               — Ojo & Babarinsa (University of New Haven / Fed. Univ. Lokoja)
#
#  Paper reference:
#    Brandes, U. (2001). A faster algorithm for betweenness centrality.
#    Journal of Mathematical Sociology, 25(2), 163–177.
#
#  Paper context (Section 2 — Graph Models Implementation):
#    "We compute the degree centrality, betweenness centrality using
#     Brandes' algorithm (Brandes, 2001) implemented on Python (numpy,
#     pandas), and clustering coefficient for detecting key nodes and
#     communities through the greedy modularity algorithm on a synthetic
#     dataset. The data was collected from 10 commercial banks at hourly
#     intervals over 41 days (starting from January 1st, 2023), where
#     there are 1000 transactions between 100 users."
#
#  Paper result (Table 1 — Top nodes with betweenness centrality):
#    Node 23 → 0.195  |  Node 47 → 0.183  |  Node 5  → 0.175
#    Node 12 → 0.172  |  Node 33 → 0.169
#
#  These high-betweenness nodes represent major intermediaries in
#  transaction chains, serving as potential liquidity hubs or risk nexuses
#  (Section 3.1). Their compromise or use in fraudulent transactions may
#  enhance risk propagation throughout the platform.
#
#  Dependencies: numpy, pandas, matplotlib, networkx (validation only)
#  Install    : pip install numpy pandas matplotlib networkx
# ============================================================================

import csv
import math
import numpy as np
import pandas as pd
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import matplotlib.patches as mpatches
from collections import defaultdict, deque


# ============================================================================
#  SECTION 1 — LOAD AND PREPARE THE TRANSACTION NETWORK
# ============================================================================

def load_transaction_graph(filepath: str) -> dict:
    """
    Load Transaction_with_banks.csv and build a directed, weighted graph.

    Graph representation (as described in the paper, Section 2):
        G = (V, E)
        V = set of user accounts (sender_id, receiver_id)
        E = set of directed transactions (sender → receiver)
```

```
Adjacency matrix entry:
   a_ij = w_ij  if a transaction exists from node i to node j
        = 0      otherwise
where w_ij is the cumulative transaction amount (weight).

Returns
-------
graph : dict with keys:
   'adj'       – dict[node] → dict[neighbour] → {'weight', 'fraud', 'freq'}
   'nodes'     – sorted list of all node IDs
   'n'         – number of nodes
   'df'        – raw DataFrame
   'fraud_set' – set of nodes that appear in fraudulent transactions
"""
with open(filepath, newline='') as f:
   rows = list(csv.DictReader(f))

df = pd.DataFrame(rows)
df['sender_id']   = df['sender_id'].astype(int)
df['receiver_id'] = df['receiver_id'].astype(int)
df['amount']      = df['amount (N)'].astype(float)
df['Is_fraud']    = df['Is_fraudulent'].astype(int)
df['freq']        = df['transaction_frequency'].astype(int)

# Build adjacency list: directed, weights = cumulative amount
adj = defaultdict(lambda: defaultdict(lambda: {'weight': 0.0,
                                               'fraud':  0,
                                               'freq':   0}))
for _, row in df.iterrows():
   s, r = row['sender_id'], row['receiver_id']
   adj[s][r]['weight'] += row['amount']
   adj[s][r]['fraud']  += row['Is_fraud']
   adj[s][r]['freq']   += 1

all_nodes = sorted(
   set(df['sender_id'].tolist() + df['receiver_id'].tolist())
)
fraud_set = set(
   df[df['Is_fraud'] == 1]['sender_id'].tolist() +
   df[df['Is_fraud'] == 1]['receiver_id'].tolist()
)

# Ensure every node appears in adj (even isolates)
for node in all_nodes:
   if node not in adj:
      adj[node] = {}

print(f"Graph loaded: {len(all_nodes)} nodes, "
     f"{sum(len(v) for v in adj.values())} directed edges, "
     f"{len(df)} transactions, "
     f"{int(df['Is_fraud'].sum())} fraudulent")

return {
   'adj':       dict(adj),
   'nodes':     all_nodes,
   'n':         len(all_nodes),
   'df':        df,
   'fraud_set': fraud_set,
```

```
    }


# ================================================================================
#  SECTION 2 — BRANDES' ALGORITHM (UNWEIGHTED, DIRECTED)
# ================================================================================

def brandes_unweighted(graph: dict) -> dict:
    """
    Brandes' O(VE) algorithm for betweenness centrality on a directed graph.

    Mathematical definition (Brandes, 2001):
        CB(v) = Σ_{s ≠ v ≠ t}  σ(s,t|v) / σ(s,t)

    where:
        σ(s,t)    = number of shortest paths from s to t
        σ(s,t|v)  = number of those paths passing through v
        CB(v)     = raw betweenness centrality of node v

    Normalisation (for directed graphs with n nodes):
        CB_norm(v) = CB(v) / [(n-1)(n-2)]

    The algorithm avoids explicit enumeration of all shortest paths by
    using a dependency accumulation trick via a stack-based back-propagation.

    Complexity: O(V · E)  — far superior to naive O(V³) enumeration.

    Algorithm steps for each source node s:
        1. BFS to compute σ(s,t) and d(s,t) for all t          [Forward pass]
        2. Back-propagate dependency δ_s(v) via stack          [Backward pass]
        3. Accumulate: CB(v) += δ_s(v)  for all v ≠ s

    Parameters
    ----------
    graph : dict returned by load_transaction_graph()

    Returns
    -------
    betweenness : dict[node_id] → normalised betweenness centrality score
    """
    nodes = graph['nodes']
    adj   = graph['adj']
    n     = graph['n']

    # Initialise betweenness scores to zero for every node
    CB = {v: 0.0 for v in nodes}

    for s in nodes:
        # ── INITIALISE BFS DATA STRUCTURES ──────────────────────────────────
        # stack S  : nodes in order of non-decreasing distance from s
        # pred[w]  : list of predecessors of w on shortest paths from s
        # sigma[v] : number of shortest paths from s to v (σ(s,v))
        # dist[v]  : shortest-path distance from s to v (-1 = unvisited)
        # delta[v] : dependency of s on v (δ_s(v)), initialised to 0

        S     = []                       # stack (finished nodes)
        pred  = {v: [] for v in nodes}   # predecessors
        sigma = {v: 0   for v in nodes}  # path counts
```

```
    dist   = {v: -1  for v in nodes}     # distances
    delta  = {v: 0.0 for v in nodes}     # pair dependencies

    sigma[s] = 1
    dist[s]  = 0

    # ── FORWARD PASS: BFS from source s ──────────────────────────────────
    # Standard BFS on directed graph (unweighted = unit edge lengths)
    queue = deque([s])

    while queue:
        v = queue.popleft()
        S.append(v)                  # v is now finalised

        for w in adj.get(v, {}):         # for each neighbour w of v
            # First time we reach w?
            if dist[w] < 0:
                queue.append(w)
                dist[w] = dist[v] + 1    # BFS distance

            # Is this a shortest path to w via v?
            if dist[w] == dist[v] + 1:
                sigma[w] += sigma[v]     # accumulate path count
                pred[w].append(v)        # v is a predecessor of w

    # ── BACKWARD PASS: dependency accumulation ──────────────────────────
    # Process nodes in reverse BFS order (from S's top)
    # Recurrence (Brandes, 2001, Eq. 7):
    #   δ_s(v) = Σ_{w: v ∈ pred(w)}  [σ(s,v)/σ(s,w)] · (1 + δ_s(w))
    while S:
        w = S.pop()
        for v in pred[w]:
            if sigma[w] > 0:             # guard against division by zero
                delta[v] += (sigma[v] / sigma[w]) * (1.0 + delta[w])

        # Accumulate into global betweenness (exclude source node s)
        if w != s:
            CB[w] += delta[w]

# ── NORMALISE ──────────────────────────────────────────────────────────
# For directed graphs: divide by (n-1)(n-2)
# For undirected: divide by (n-1)(n-2)/2
norm = (n - 1) * (n - 2) if n > 2 else 1.0
betweenness = {v: CB[v] / norm for v in nodes}

return betweenness


# ============================================================================
#  SECTION 3 — BRANDES' ALGORITHM (WEIGHTED, DIRECTED) — DIJKSTRA VARIANT
# ============================================================================

def brandes_weighted(graph: dict, weight_key: str = 'weight') -> dict:
    """
    Weighted variant of Brandes' algorithm using Dijkstra's shortest-path.

    Used when edge weights represent transaction amounts (as in the paper's
```

```
multilayer Layer 1: transaction amount). Shortest paths are defined by
minimum cumulative weight — here we invert weights so that higher-amount
transaction paths are preferred (a larger amount = shorter effective
distance), capturing the paper's notion that high-value nodes are hubs.

Alternatively, use raw weights as distances for minimum-cost routing.
Both modes are supported via the `invert` parameter in the caller.

Complexity: O(V · (E + V) log V)  using a priority queue.

Parameters
----------
graph      : dict returned by load_transaction_graph()
weight_key : edge attribute to use as weight ('weight', 'freq', 'fraud')

Returns
-------
betweenness : dict[node_id] → normalised weighted betweenness centrality
"""
import heapq

nodes = graph['nodes']
adj   = graph['adj']
n     = graph['n']

CB = {v: 0.0 for v in nodes}

for s in nodes:
    # ── INITIALISE ──────────────────────────────────────────────────────────
    S     = []
    pred  = {v: [] for v in nodes}
    sigma = {v: 0.0 for v in nodes}
    dist  = {v: math.inf for v in nodes}
    delta = {v: 0.0 for v in nodes}

    sigma[s] = 1.0
    dist[s]  = 0.0

    # Min-heap: (distance, node)
    heap = [(0.0, s)]

    # ── FORWARD PASS: Dijkstra from source s ────────────────────────────────
    while heap:
        d_v, v = heapq.heappop(heap)

        if d_v > dist[v]:
            continue                  # stale entry; skip

        S.append(v)

        for w, edge_data in adj.get(v, {}).items():
            # Edge weight: use inverse amount so high-value = preferred
            raw_w = edge_data.get(weight_key, 1.0)
            edge_weight = 1.0 / raw_w if raw_w > 0 else 1.0

            new_dist = dist[v] + edge_weight

            # Found shorter path to w?
```

```
            if new_dist < dist[w]:
                dist[w] = new_dist
                sigma[w] = 0.0
                pred[w]  = []
                heapq.heappush(heap, (new_dist, w))

            # Found equally short path to w via v?
            if math.isclose(new_dist, dist[w], rel_tol=1e-9):
                sigma[w] += sigma[v]
                pred[w].append(v)

        # ── BACKWARD PASS ──────────────────────────────────────────────
        while S:
            w = S.pop()
            for v in pred[w]:
                if sigma[w] > 0:
                    delta[v] += (sigma[v] / sigma[w]) * (1.0 + delta[w])
            if w != s:
                CB[w] += delta[w]

    # Normalise
    norm = (n - 1) * (n - 2) if n > 2 else 1.0
    betweenness = {v: CB[v] / norm for v in nodes}

    return betweenness


# ============================================================================
#  SECTION 4 — COMPLEMENTARY CENTRALITY MEASURES
# ============================================================================

def degree_centrality(graph: dict) -> tuple[dict, dict, dict]:
    """
    Compute in-degree, out-degree, and normalised degree centrality.

    In-degree  centrality: fraction of nodes that send TO node v
    Out-degree centrality: fraction of nodes that v sends TO

    Used in Section 3.1 of the paper:
      "High out-degree nodes identify repeat senders in transactions.
       Nodes with high in-degree signal aggregation activities, wherein
       several parties are directing funds to an account."
    """
    nodes = graph['nodes']
    adj   = graph['adj']
    n     = graph['n']

    out_deg = {v: len(adj.get(v, {})) for v in nodes}
    in_deg  = {v: 0 for v in nodes}
    for v in nodes:
        for w in adj.get(v, {}):
            in_deg[w] = in_deg.get(w, 0) + 1

    norm = (n - 1) if n > 1 else 1.0
    dc_in  = {v: in_deg[v]  / norm for v in nodes}
    dc_out = {v: out_deg[v] / norm for v in nodes}
    dc_total = {v: (in_deg[v] + out_deg[v]) / (2 * norm) for v in nodes}
```

```
    return dc_in, dc_out, dc_total


def clustering_coefficient(graph: dict) -> dict:
    """
    Compute the local clustering coefficient for each node.

    For directed graphs, the clustering coefficient of node v is:
        C(v) = |{(u,w) : u,w ∈ N(v), (u,w) ∈ E or (w,u) ∈ E}|
               ÷ [k(v) · (k(v) − 1)]

    where N(v) is the combined neighbourhood (in + out neighbours),
    k(v) = |N(v)|.

    Cited in the paper (Section 2):
      "We compute ... clustering coefficient for detecting key nodes
       and communities."
    """
    nodes = graph['nodes']
    adj   = graph['adj']

    cc = {}
    for v in nodes:
        # Combined neighbourhood (in + out), excluding v itself
        out_neighbours = set(adj.get(v, {}).keys())
        in_neighbours  = set()
        for u in nodes:
            if v in adj.get(u, {}):
                in_neighbours.add(u)
        neighbours = (out_neighbours | in_neighbours) - {v}
        k = len(neighbours)

        if k < 2:
            cc[v] = 0.0
            continue

        triangles = 0
        nbrs = list(neighbours)
        for i in range(len(nbrs)):
            for j in range(i + 1, len(nbrs)):
                u, w = nbrs[i], nbrs[j]
                # Check both directed edges
                if w in adj.get(u, {}) or u in adj.get(w, {}):
                    triangles += 1

        cc[v] = triangles / (k * (k - 1))

    return cc


def weighted_degree(graph: dict) -> dict:
    """
    Compute weighted (strength) degree: sum of all edge weights per node.
    High strength = high transaction volume = potential hub or risk nexus.
    """
    nodes = graph['nodes']
    adj   = graph['adj']
```

```
    strength = {}
    for v in nodes:
        out_w = sum(d['weight'] for d in adj.get(v, {}).values())
        in_w  = sum(
            adj[u][v]['weight']
            for u in nodes if v in adj.get(u, {})
        )
        strength[v] = out_w + in_w

    return strength


# ============================================================================
#  SECTION 5 — VALIDATION AGAINST PAPER'S TABLE 1
# ============================================================================

def validate_against_paper(betweenness: dict) -> None:
    """
    Compare computed betweenness centrality against Table 1 of the paper.

    Paper's Table 1 (Top nodes with betweenness centrality):
        Node 23 → 0.195
        Node 47 → 0.183
        Node 5  → 0.175
        Node 12 → 0.172
        Node 33 → 0.169

    Note: The paper uses Brandes' algorithm on the full 1,000-transaction
    directed graph. Minor floating-point differences are expected; rank
    order is the key validation criterion.
    """
    paper_results = {23: 0.195, 47: 0.183, 5: 0.175, 12: 0.172, 33: 0.169}

    # Get top-5 from our computation
    top5 = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)[:5]

    print("\n" + "=" * 65)
    print("VALIDATION — Comparing with Paper Table 1")
    print("=" * 65)
    print(f"{'Rank':<6} {'Node':>6} {'Computed BC':>14} {'Paper BC':>12} {'Delta':>10} {'Match':>8}")
    print("-" * 65)

    paper_top5_nodes = list(paper_results.keys())
    computed_top5_nodes = [n for n, _ in top5]

    for rank, (node, score) in enumerate(top5, 1):
        paper_val = paper_results.get(node, None)
        delta     = f"{abs(score - paper_val):.4f}" if paper_val else "  N/A  "
        match     = "✓" if paper_val and abs(score - paper_val) < 0.05 else "~"
        paper_str = f"{paper_val:.3f}" if paper_val else "  —  "
        print(f"{rank:<6} {node:>6} {score:>14.4f} {paper_str:>12} {delta:>10} {match:>8}")

    print("-" * 65)
    # Rank-order correlation
    rank_match = sum(1 for i, (n, _) in enumerate(top5)
                     if i < len(paper_top5_nodes) and n == paper_top5_nodes[i])
    print(f"Rank-order matches (top-5): {rank_match}/5")
```

```
    print(f"Node set overlap (top-5)  : "
          f"{len(set(computed_top5_nodes) & set(paper_top5_nodes))}/5")
    print("=" * 65)


# ============================================================================
#  SECTION 6 — FRAUD RISK SCORING
# ============================================================================

def compute_fraud_risk_score(
    graph: dict,
    betweenness: dict,
    dc_in: dict,
    dc_out: dict,
    cc: dict,
    strength: dict
) -> pd.DataFrame:
    """
    Compute a composite fraud risk score for each node combining:
       1. Normalised betweenness centrality  (structural intermediary role)
       2. In-degree centrality               (aggregation / mule receiver)
       3. Out-degree centrality              (smurfing / layering sender)
       4. Clustering coefficient             (tight fraud ring membership)
       5. Fraud transaction rate             (direct fraud signal from data)

    Score formula:
       FRS(v) = 0.30 · BC_norm(v)
              + 0.20 · DC_in_norm(v)
              + 0.15 · DC_out_norm(v)
              + 0.15 · [1 − CC(v)]        (low CC = broker, not ring)
              + 0.20 · fraud_rate(v)

    Weights reflect the paper's emphasis on betweenness centrality as
    the primary structural fraud indicator (Section 3.1 and 4).
    """
    nodes = graph['nodes']
    df    = graph['df']

    # Per-node fraud rate from transaction data
    fraud_rate = {}
    for node in nodes:
        sent = df[df['sender_id'] == node]
        recv = df[df['receiver_id'] == node]
        total = len(sent) + len(recv)
        frauds = int(sent['Is_fraud'].sum()) + int(recv['Is_fraud'].sum())
        fraud_rate[node] = frauds / total if total > 0 else 0.0

    # Normalise each component to [0, 1]
    def norm_dict(d):
        vals = list(d.values())
        mx, mn = max(vals), min(vals)
        rng = mx - mn if mx != mn else 1.0
        return {k: (v - mn) / rng for k, v in d.items()}

    bc_n   = norm_dict(betweenness)
    din_n  = norm_dict(dc_in)
    dout_n = norm_dict(dc_out)
    cc_n   = norm_dict(cc)
```

```
    fr_n   = norm_dict(fraud_rate)

    records = []
    for node in nodes:
        frs = (0.30 * bc_n[node]
             + 0.20 * din_n[node]
             + 0.15 * dout_n[node]
             + 0.15 * (1 - cc_n[node])
             + 0.20 * fr_n[node])
        records.append({
            'node_id':          node,
            'betweenness_bc':   round(betweenness[node], 5),
            'in_degree_cent':   round(dc_in[node], 5),
            'out_degree_cent':  round(dc_out[node], 5),
            'clustering_coeff': round(cc[node], 5),
            'strength':         round(strength[node], 2),
            'fraud_rate':       round(fraud_rate[node], 4),
            'fraud_risk_score': round(frs, 5),
            'is_high_risk':     frs >= 0.5,
        })

    result_df = pd.DataFrame(records).sort_values(
        'fraud_risk_score', ascending=False).reset_index(drop=True)

    return result_df


# ============================================================================
#  SECTION 7 — VISUALISATION
# ============================================================================

def visualise_results(graph: dict, betweenness: dict,
                      betweenness_w: dict, risk_df: pd.DataFrame) -> None:
    """
    Generate a 4-panel figure replicating and extending Figure 1 of the paper:
        A. Transaction network with node size ∝ betweenness centrality
        B. Betweenness centrality distribution (top-20 nodes)
        C. Unweighted vs weighted betweenness comparison
        D. Fraud risk score for top-20 highest-risk nodes
    """
    import networkx as nx

    # --- build networkx graph for layout ----------------------------------
    adj   = graph['adj']
    nodes = graph['nodes']
    df    = graph['df']

    G = nx.DiGraph()
    for v in nodes:
        G.add_node(v)
    for v, nbrs in adj.items():
        for w, data in nbrs.items():
            G.add_edge(v, w, weight=data['weight'],
                       fraud=data['fraud'], freq=data['freq'])

    pos = nx.spring_layout(G, seed=42, k=0.45)

    # ── colour / size palettes ──────────────────────────────────────────────
```

```
NAVY    = '#1B2A4A'
TEAL    = '#1A7A8A'
CRIMSON = '#C0392B'
AMBER   = '#E67E22'
GREEN   = '#27AE60'
CREAM   = '#FDFBF7'
LGREY   = '#ECF0F1'

paper_top5 = {23, 47, 5, 12, 33}   # from Table 1 of the paper
fraud_set  = graph['fraud_set']

fig = plt.figure(figsize=(18, 14), facecolor=CREAM)
fig.suptitle(
   "Brandes' Betweenness Centrality — Nigerian Bank Transaction Network\n"
   "Ojo & Babarinsa | 1,000 transactions, 100 nodes, 10 banks, 41 days",
   fontsize=15, fontweight='bold', color=NAVY, y=0.99
)
gs = gridspec.GridSpec(2, 2, figure=fig,
                       hspace=0.42, wspace=0.32,
                       left=0.06, right=0.97,
                       top=0.92, bottom=0.06)

# ── Panel A: Transaction network (Figure 1 of paper) ──────────────────
ax1 = fig.add_subplot(gs[0, 0])
ax1.set_facecolor(LGREY)

bc_vals  = np.array([betweenness[n] for n in G.nodes()])
bc_max   = bc_vals.max() if bc_vals.max() > 0 else 1.0
ns       = [200 + 2500 * (betweenness[n] / bc_max) for n in G.nodes()]
nc       = [CRIMSON if n in paper_top5 else
            '#E8A0A0' if n in fraud_set else TEAL
            for n in G.nodes()]
ec       = ['#C0392B' if d.get('fraud', 0) > 0 else '#95A5A6'
            for _, _, d in G.edges(data=True)]

nx.draw_networkx_edges(G, pos, ax=ax1, alpha=0.3,
                       edge_color=ec, arrows=True, arrowsize=6)
nx.draw_networkx_nodes(G, pos, ax=ax1, node_color=nc,
                       node_size=ns, alpha=0.88)

# Label only paper's top-5 nodes
labels = {n: str(n) for n in paper_top5 if n in G.nodes()}
nx.draw_networkx_labels(G, pos, labels=labels, ax=ax1,
                        font_size=8, font_color='white',
                        font_weight='bold')

ax1.set_title("A.  Transaction Network\n"
              "(node size ∝ betweenness; red = paper Table 1 nodes)",
              fontsize=10, fontweight='bold', color=NAVY)
legend_elems = [
   mpatches.Patch(color=CRIMSON, label='Top-5 BC nodes (Table 1)'),
   mpatches.Patch(color='#E8A0A0', label='Fraud-linked node'),
   mpatches.Patch(color=TEAL, label='Normal node'),
]
ax1.legend(handles=legend_elems, fontsize=8, loc='lower right')
ax1.axis('off')

# ── Panel B: Betweenness centrality bar chart (top-20) ────────────────
```

```
ax2 = fig.add_subplot(gs[0, 1])
ax2.set_facecolor(LGREY)

top20 = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)[:20]
t20_nodes = [str(n) for n, _ in top20]
t20_bc    = [v for _, v in top20]
colours20 = [CRIMSON if n in paper_top5 else
             AMBER if n in fraud_set else TEAL
             for n, _ in top20]

bars = ax2.barh(t20_nodes[::-1], t20_bc[::-1],
                color=colours20[::-1], edgecolor='white',
                linewidth=0.8)
# Add value labels
for bar, val in zip(bars, t20_bc[::-1]):
    ax2.text(bar.get_width() + 0.001, bar.get_y() + bar.get_height()/2,
             f'{val:.4f}', va='center', fontsize=7.5, color=NAVY)

# Mark paper's reported values
paper_vals = {23: 0.195, 47: 0.183, 5: 0.175, 12: 0.172, 33: 0.169}
for i, (node, _) in enumerate(reversed(top20)):
    if node in paper_vals:
        ax2.axvline(paper_vals[node], color=NAVY, linestyle=':',
                    linewidth=0.8, alpha=0.6)

ax2.set_xlabel('Normalised Betweenness Centrality', fontsize=10, color=NAVY)
ax2.set_title("B.  Betweenness Centrality — Top 20 Nodes\n"
              "(red = paper Table 1 nodes; dotted = paper-reported value)",
              fontsize=10, fontweight='bold', color=NAVY)
legend_b = [
    mpatches.Patch(color=CRIMSON, label='Table 1 nodes'),
    mpatches.Patch(color=AMBER,   label='Other fraud nodes'),
    mpatches.Patch(color=TEAL,    label='Normal nodes'),
]
ax2.legend(handles=legend_b, fontsize=8, loc='lower right')
ax2.grid(axis='x', alpha=0.4)
ax2.tick_params(colors=NAVY, labelsize=8)

# ── Panel C: Unweighted vs weighted betweenness ──────────────────────
ax3 = fig.add_subplot(gs[1, 0])
ax3.set_facecolor(LGREY)

top15_uw = dict(sorted(betweenness.items(),
                       key=lambda x: x[1], reverse=True)[:15])
top15_w  = {n: betweenness_w[n] for n in top15_uw}

x15 = np.arange(len(top15_uw))
w = 0.38
b1 = ax3.bar(x15 - w/2, list(top15_uw.values()), w,
             label='Unweighted BC', color=TEAL, edgecolor='white', linewidth=0.8)
b2 = ax3.bar(x15 + w/2, list(top15_w.values()),  w,
             label='Weighted BC (amount)', color=AMBER, edgecolor='white', linewidth=0.8)

ax3.set_xticks(x15)
ax3.set_xticklabels([f'N{n}' for n in top15_uw.keys()],
                    rotation=45, ha='right', fontsize=8)
ax3.set_ylabel('Betweenness Centrality', fontsize=10, color=NAVY)
ax3.set_title("C.  Unweighted vs Weighted Betweenness\n"
```

```
            "(weighted = Dijkstra on inverse transaction amounts)",
            fontsize=10, fontweight='bold', color=NAVY)
    ax3.legend(fontsize=9)
    ax3.grid(axis='y', alpha=0.4)
    ax3.tick_params(colors=NAVY)

    # ── Panel D: Fraud risk score for top-20 ──────────────────────────────
    ax4 = fig.add_subplot(gs[1, 1])
    ax4.set_facecolor(LGREY)

    top20_risk = risk_df.head(20)
    colours_r  = [CRIMSON if n in paper_top5 else
                  AMBER   if hr else TEAL
                  for n, hr in zip(top20_risk['node_id'],
                                   top20_risk['is_high_risk'])]

    bars_r = ax4.bar(range(len(top20_risk)),
                     top20_risk['fraud_risk_score'],
                     color=colours_r, edgecolor='white', linewidth=0.8)
    ax4.axhline(0.5, color=NAVY, linestyle='--', linewidth=1.5,
                label='High-risk threshold (0.5)')

    for i, (_, row) in enumerate(top20_risk.iterrows()):
        ax4.text(i, row['fraud_risk_score'] + 0.005,
                 str(row['node_id']), ha='center', va='bottom',
                 fontsize=7, color=NAVY, fontweight='bold')

    ax4.set_xlabel('Node (ranked by fraud risk score)', fontsize=10, color=NAVY)
    ax4.set_ylabel('Composite Fraud Risk Score', fontsize=10, color=NAVY)
    ax4.set_title("D.  Composite Fraud Risk Score — Top 20 Nodes\n"
                  "(BC 30% + in-deg 20% + out-deg 15% + 1−CC 15% + fraud-rate 20%)",
                  fontsize=10, fontweight='bold', color=NAVY)
    ax4.legend(fontsize=9)
    ax4.set_ylim(0, 0.85)
    ax4.grid(axis='y', alpha=0.4)
    ax4.tick_params(colors=NAVY, labelsize=8)

    plt.savefig('/mnt/user-data/outputs/brandes_centrality_analysis.png',
                dpi=160, bbox_inches='tight', facecolor=CREAM)
    plt.close()
    print("Figure saved → brandes_centrality_analysis.png")


# ═══════════════════════════════════════════════════════════════════════════
#  SECTION 8 — PRINT REPORT
# ═══════════════════════════════════════════════════════════════════════════

def print_full_report(betweenness: dict, betweenness_w: dict,
                      dc_in: dict, dc_out: dict,
                      cc: dict, strength: dict,
                      risk_df: pd.DataFrame, graph: dict) -> None:

    nodes = graph['nodes']
    n     = graph['n']

    print("\n" + "=" * 65)
    print("BRANDES' ALGORITHM — FULL RESULTS REPORT")
    print(f"Graph: {n} nodes, 100 users, 1,000 transactions, 10 banks")
```

```
    print("=" * 65)

    # ── Top 10: Unweighted betweenness ──────────────────────────────────────
    print("\n── Top 10: Unweighted Betweenness Centrality (Brandes, 2001)")
    print(f"{'Rank':<6}{'Node':>6}{'BC Score':>14}{'In-Deg':>10}"
          f"{'Out-Deg':>10}{'CC':>10}{'Fraud Rate':>12}")
    print("-" * 65)
    top10 = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)[:10]
    for rank, (node, bc) in enumerate(top10, 1):
        print(f"{rank:<6}{node:>6}{bc:>14.5f}"
              f"{dc_in[node]:>10.4f}{dc_out[node]:>10.4f}"
              f"{cc[node]:>10.4f}"
              f"{risk_df.loc[risk_df['node_id']==node,'fraud_rate'].values[0]:>12.4f}")

    # ── Top 10: Weighted betweenness ────────────────────────────────────────
    print("\n── Top 10: Weighted Betweenness Centrality (Dijkstra variant)")
    print(f"{'Rank':<6}{'Node':>6}{'Weighted BC':>14}{'Strength (N)':>15}")
    print("-" * 42)
    top10w = sorted(betweenness_w.items(), key=lambda x: x[1], reverse=True)[:10]
    for rank, (node, wbc) in enumerate(top10w, 1):
        print(f"{rank:<6}{node:>6}{wbc:>14.5f}{strength[node]:>15,.2f}")

    # ── Top 10: Fraud risk score ────────────────────────────────────────────
    print("\n── Top 10: Composite Fraud Risk Score")
    print(f"{'Rank':<6}{'Node':>6}{'FRS':>10}{'BC':>10}"
          f"{'In-Deg':>10}{'FR':>10}{'High-Risk':>12}")
    print("-" * 65)
    for rank, row in risk_df.head(10).iterrows():
        print(f"{rank+1:<6}{int(row['node_id']):>6}{row['fraud_risk_score']:>10.4f}"
              f"{row['betweenness_bc']:>10.5f}{row['in_degree_cent']:>10.4f}"
              f"{row['fraud_rate']:>10.4f}{'YES' if row['is_high_risk'] else 'no':>12}")

    # ── Network-level statistics ────────────────────────────────────────────
    bc_vals = list(betweenness.values())
    print("\n── Network-Level Betweenness Statistics")
    print(f"  Mean BC          : {np.mean(bc_vals):.6f}")
    print(f"  Std Dev BC       : {np.std(bc_vals):.6f}")
    print(f"  Max BC (most hub) : {max(bc_vals):.6f}  → Node "
          f"{max(betweenness, key=betweenness.get)}")
    print(f"  Min BC           : {min(bc_vals):.6f}")
    print(f"  Nodes BC > mean   : "
          f"{sum(1 for v in bc_vals if v > np.mean(bc_vals))}")
    print(f"  High-risk nodes   : {risk_df['is_high_risk'].sum()}")
    print("=" * 65)


# ============================================================================
#  MAIN EXECUTION
# ============================================================================

if __name__ == '__main__':

    CSV_PATH = '/mnt/user-data/uploads/Transaction_with_banks.csv'

    print("=" * 65)
    print("  BRANDES' BETWEENNESS CENTRALITY")
    print("  Nigerian Bank Fraud Detection — Ojo & Babarinsa")
    print("=" * 65)
```

```
# 1. Load data and build graph
print("\n[1/6] Loading transaction network...")
graph = load_transaction_graph(CSV_PATH)

# 2. Run Brandes' algorithm (unweighted)
print("\n[2/6] Running Brandes' unweighted algorithm [O(VE)]...")
betweenness = brandes_unweighted(graph)
print(f"      Done. Top node: {max(betweenness, key=betweenness.get)} "
      f"(BC = {max(betweenness.values()):.5f})")

# 3. Run Brandes' algorithm (weighted — Dijkstra variant)
print("\n[3/6] Running Brandes' weighted algorithm [O(V(E+V)logV)]...")
betweenness_w = brandes_weighted(graph, weight_key='weight')
print(f"      Done. Top node: {max(betweenness_w, key=betweenness_w.get)} "
      f"(weighted BC = {max(betweenness_w.values()):.5f})")

# 4. Compute complementary centrality measures
print("\n[4/6] Computing degree centrality and clustering coefficients...")
dc_in, dc_out, dc_total = degree_centrality(graph)
cc       = clustering_coefficient(graph)
strength = weighted_degree(graph)

# 5. Validate against paper's Table 1
validate_against_paper(betweenness)

# 6. Build composite fraud risk scores
print("\n[5/6] Computing composite fraud risk scores...")
risk_df = compute_fraud_risk_score(
    graph, betweenness, dc_in, dc_out, cc, strength
)

# 7. Print full report
print_full_report(betweenness, betweenness_w, dc_in, dc_out,
                  cc, strength, risk_df, graph)

# 8. Save risk dataframe to CSV
risk_df.to_csv('/mnt/user-data/outputs/node_risk_scores.csv', index=False)
print("\nNode risk scores saved → node_risk_scores.csv")

# 9. Generate visualisation
print("\n[6/6] Generating visualisation...")
visualise_results(graph, betweenness, betweenness_w, risk_df)

print("\nAll outputs written to /mnt/user-data/outputs/")
print("Done.")
```

APPENDIX B

Bank	sender_id	receiver_id	amount (N)	fraud_rate	total_fraudulent	transaction_frequency	KYC_level	Is_fraudulent	cycles_total	mule_score
GTB	52	34	111.8	0.37	3	2	1	0	1271	0.372
GTB	93	8	575.2	0.95	2	8	2	0	939	0.285
GTB	15	40	28.18	0.73	4	8	1	1	567	0.28059
GTB	72	83	487.5	0.6	0	17	1	0	646	0.27667
GTB	61	42	218.3	0.16	4	15	3	1	957	0.23482
GTB	21	41	352.6	0.16	0	17	3	0	1458	0.225
GTB	83	6	778.1	0.06	1	13	1	0	385	0.17833
GTB	87	52	941.8	0.87	0	19	2	0	597	0.14333
GTB	75	26	81.69	0.6	3	5	1	0	845	0.14026
GTB	75	64	538.9	0.71	2	12	1	0	366	0.14
GTB	88	98	883.7	0.02	1	1	2	0	1437	0.11403
GTB	100	59	91.68	0.97	0	5	3	0	224	0.105
GTB	24	56	826.3	0.83	4	8	1	0	609	0.10294
GTB	3	59	364.2	0.21	1	13	3	0	600	0.09333
GTB	22	70	328.9	0.18	1	8	3	0	747	0.09333
GTB	53	33	977.7	0.18	0	11	2	1	768	0.09211
GTB	2	53	208.9	0.3	1	1	2	0	792	0.09211
GTB	88	22	696.3	0.52	4	13	3	0	834	0.0875
GTB	30	21	823.2	0.43	1	19	1	0	166	0.0875
GTB	38	70	50.77	0.29	4	3	3	0	925	0.0875
GTB	2	70	673.7	0.61	0	17	1	0	1337	0.08448
GTB	64	4	952	0.14	2	5	1	0	665	0.08235

GTB	60	94	133	0.29	1	8	2	0	1166	0.08077
GTB	21	75	897.3	0.37	2	14	1	0	391	0.08077
GTB	33	62	598.8	0.46	4	4	1	0	819	0.07955
GTB	76	62	620.4	0.79	1	11	3	0	542	0.07778
GTB	58	94	616.8	0.2	3	15	2	0	700	0.07609
GTB	22	95	305.9	0.51	2	19	1	0	1351	0.075
GTB	89	24	935.9	0.59	1	7	1	0	837	0.07368
GTB	49	55	947.9	0.05	2	19	2	0	1079	0.07292
GTB	91	9	629.1	0.61	1	19	3	0	1704	0.07241
GTB	59	3	785.6	0.17	0	11	2	0	482	0.07
GTB	42	31	616.1	0.07	2	5	2	0	1734	0.06731
GTB	92	40	489.6	0.95	3	5	1	0	1470	0.06731
GTB	60	36	664.2	0.97	3	8	1	0	672	0.06667
GTB	80	24	557.5	0.81	4	10	3	0	973	0.06667
GTB	15	95	601.5	0.3	1	6	1	0	563	0.06562
GTB	62	6	786	0.1	2	15	1	0	899	0.06364
GTB	62	66	486.1	0.68	4	16	2	0	918	0.06364
GTB	47	84	50.35	0.44	2	17	1	0	616	0.06176
GTB	62	92	172.4	0.12	1	8	3	0	738	0.05833
GTB	51	75	444.2	0.5	3	8	2	0	538	0.05833
GTB	55	4	727.6	0.03	0	9	3	0	705	0.05833
GTB	64	79	597.4	0.91	3	2	2	1	569	0.05833
GTB	3	6	653.8	0.26	4	15	1	0	1048	0.05833
GTB	51	94	484.5	0.66	0	18	2	0	1379	0.056
GTB	7	51	619.6	0.31	4	3	2	0	1191	0.056

GTB	21	62	335.7	0.52	1	17	1	0	525	0.05526
GTB	73	57	475.5	0.55	0	4	1	0	594	0.05526
GTB	39	66	746.8	0.18	4	4	3	0	331	0.05385
GTB	18	79	873.1	0.97	1	14	3	0	843	0.05
GTB	4	75	439.3	0.78	3	9	1	0	865	0.04773
GTB	89	8	53.23	0.94	1	9	1	0	1067	0.04773
GTB	60	26	170.1	0.89	2	18	1	1	749	0.04773
GTB	14	51	30.65	0.6	1	10	2	1	957	0.04773
GTB	9	45	656.7	0.92	1	4	2	0	329	0.04667
GTB	90	44	339	0.09	2	12	2	0	1068	0.04565
GTB	53	5	547.1	0.2	4	8	1	0	862	0.04375
GTB	2	70	498.3	0.05	4	2	3	0	1207	0.04375
GTB	84	26	90.71	0.33	1	13	1	0	1103	0.04375
GTB	92	68	409.2	0.39	2	16	2	0	486	0.04375
GTB	60	19	228.4	0.27	2	2	2	0	1117	0.04375
GTB	71	84	424.2	0.83	3	11	1	1	770	0.042
GTB	44	97	211.5	0.36	4	10	1	0	1066	0.042
GTB	8	20	842.1	0.28	0	15	2	0	536	0.04118
GTB	47	12	182.5	0.54	4	17	2	0	487	0.04118
GTB	35	47	961.2	0.14	4	8	2	0	471	0.03889
GTB	78	1	421.8	0.8	3	18	1	0	460	0.03889
GTB	81	90	847.8	0.07	0	9	2	0	586	0.03889
GTB	36	14	632.5	0.99	3	8	2	0	747	0.03684
GTB	50	64	738	0.77	0	14	3	0	558	0.03684
GTB	4	38	770.1	0.2	1	18	1	0	731	0.03684

GTB	2	37	509.8	0.01	0	12	3	0	672	0.03684
GTB	6	11	547.9	0.82	3	19	3	0	929	0.03333
GTB	54	100	518.9	0.71	4	14	2	0	251	0.03182
GTB	4	77	592.2	0.73	0	17	1	0	223	0.03182
GTB	54	3	39.1	0.77	0	3	2	0	772	0.03182
GTB	93	33	401.6	0.07	2	7	1	0	1041	0.03043
GTB	63	6	610.1	0.36	4	17	3	0	1189	0.03043
GTB	18	50	601.8	0.12	1	12	1	1	2062	0.03
GTB	90	10	795	0.86	3	9	2	0	362	0.025
GTB	44	5	658.5	0.62	1	15	2	0	1406	0.02414
GTB	34	23	992.2	0.33	4	18	2	0	556	0.02333
GTB	74	10	291	0.06	0	13	1	0	698	0.02059
GTB	62	44	365.6	0.31	0	8	3	0	631	0.02059
GTB	100	2	390	0.33	4	14	3	0	747	0.02059
GTB	14	13	471.2	0.73	0	8	1	0	630	0.01944
GTB	95	40	837.6	0.64	0	14	2	0	723	0.01944
GTB	48	2	243.4	0.89	0	7	3	1	785	0.01842
GTB	15	84	778.2	0.47	3	12	2	0	934	0.01842
GTB	72	65	411.3	0.12	3	11	1	0	734	0.0175
GTB	78	63	605.8	0.71	2	11	2	1	423	0.0175
GTB	87	73	636.2	0.76	1	13	3	0	841	0.01667
GTB	62	17	379.1	0.56	4	18	3	0	904	0.01591
GTB	40	9	517.5	0.77	4	2	2	0	751	0.01522
GTB	85	75	418.9	0.49	4	15	1	0	1068	0
GTB	80	15	78.44	0.52	2	18	2	0	164	0

GTB	82	24	442.8	0.43	3	15	1	0	468	0
GTB	53	38	106.4	0.03	2	19	2	0	310	0
GTB	24	35	411	0.11	4	12	3	0	532	0
UBA	26	94	597	0.03	1	13	1	0		
UBA	89	95	583.6	0.64	0	11	1	0		
UBA	60	49	666	0.31	3	5	1	0		
UBA	41	69	104.3	0.51	3	1	3	1		
UBA	29	62	655.6	0.91	2	3	2	0		
UBA	15	60	319	0.25	3	19	3	1		
UBA	45	50	405.1	0.41	4	7	1	1		
UBA	65	78	859.4	0.76	4	12	2	0		
UBA	89	75	18.94	0.23	2	15	3	0		
UBA	71	9	592.5	0.08	4	5	1	0		
UBA	9	34	602.7	0.29	1	15	1	0		
UBA	88	76	402.7	0.16	2	8	3	0		
UBA	1	99	244.7	0.93	1	10	3	0		
UBA	8	35	242.3	0.81	0	16	3	0		
UBA	88	1	884.9	0.63	1	8	1	1		
UBA	63	40	447.6	0.87	0	1	3	1		
UBA	11	64	337.6	0.8	0	12	1	0		
UBA	81	22	454.3	0.19	1	5	3	0		
UBA	8	60	672.7	0.89	2	3	1	0		
UBA	35	64	154.3	0.54	4	1	1	0		
UBA	35	93	454.9	0.81	4	4	2	0		
UBA	33	72	592	0.9	0	8	3	0		

UBA	5	11	949.1	0.32	0	3	3	0		
UBA	41	14	260	0.11	2	10	1	1		
UBA	28	60	365.7	0.23	3	2	1	0		
UBA	7	30	385.6	0.43	1	11	1	0		
UBA	73	35	609.5	0.82	0	10	2	0		
UBA	72	85	363.2	0.86	4	14	2	0		
UBA	12	37	538.2	0.01	2	7	1	0		
UBA	34	5	784	0.51	1	2	3	1		
UBA	33	83	708.3	0.42	2	9	3	0		
UBA	48	78	451.9	0.22	1	16	2	0		
UBA	23	26	865.6	0.12	1	13	2	0		
UBA	62	62	562.6	0.34	0	9	1	0		
UBA	88	4	544.5	0.94	2	4	3	0		
UBA	37	89	43.85	0.32	2	10	1	0		
UBA	99	42	986.8	0.52	4	12	2	0		
UBA	44	89	131.7	0.7	4	17	3	0		
UBA	86	18	238.8	0.36	0	14	3	0		
UBA	91	40	61.18	0.97	0	14	2	0		
UBA	35	72	643.6	0.96	3	2	1	1		
UBA	65	39	232.5	0.25	4	11	1	0		
UBA	99	14	188.2	0.5	1	16	3	0		
UBA	47	32	97.67	0.3	2	19	2	0		
UBA	78	51	112.3	0.28	0	6	3	0		
UBA	3	38	588.8	0.04	2	19	2	0		
UBA	1	97	691.3	0.61	2	5	1	0		

UBA	5	23	403.4	0.5	3	4	2	0		
UBA	90	63	485.1	0.05	4	15	1	1		
UBA	14	15	689.1	0.28	0	4	2	0		
UBA	27	97	27.21	0.91	2	10	3	1		
UBA	9	25	329.9	0.24	0	3	1	0		
UBA	79	17	973.4	0.14	1	5	3	1		
UBA	15	97	584.3	0.49	3	6	3	0		
UBA	90	66	261.7	0.99	1	8	1	0		
UBA	42	78	526.5	0.24	0	7	1	1		
UBA	77	53	349.2	0.67	0	10	3	0		
UBA	51	51	536.9	0.76	4	6	3	1		
UBA	63	39	134.5	0.24	2	14	3	0		
UBA	96	51	141.4	0.73	2	4	3	0		
UBA	52	70	152.1	0.37	1	13	1	0		
UBA	96	6	940.4	0.63	3	17	1	1		
UBA	4	67	735.6	0.63	2	7	3	0		
UBA	94	7	170.8	0.54	4	4	2	0		
UBA	23	51	196.5	0.09	1	14	2	0		
UBA	15	72	733.5	0.84	1	3	1	0		
UBA	43	42	164.6	0.32	4	5	3	0		
UBA	29	64	242.2	0.19	3	16	3	0		
UBA	36	15	963.7	0.04	4	4	1	0		
UBA	13	29	262.8	0.59	3	2	2	0		
UBA	32	33	665	0.68	4	18	2	0		
UBA	71	94	978.8	0.02	4	19	2	1		

UBA	59	27	380.6	0.51	0	19	1	0		
UBA	86	36	27.34	0.23	1	17	2	0		
UBA	28	29	251.7	0.65	4	14	1	0		
UBA	66	38	832.6	0.17	4	19	1	0		
UBA	42	57	866	0.69	2	13	3	0		
UBA	45	97	770.8	0.39	1	12	2	0		
UBA	62	27	122	0.94	2	6	1	0		
UBA	57	55	797.6	0.14	1	4	2	1		
UBA	6	33	919.5	0.34	2	19	2	0		
UBA	28	68	604.6	0.11	4	16	3	1		
UBA	28	86	941.5	0.92	4	13	2	1		
UBA	44	66	828.6	0.88	0	4	3	0		
UBA	84	10	670.3	0.26	2	17	1	0		
UBA	30	5	581.4	0.66	3	1	2	0		
UBA	62	74	355.1	0.82	0	18	1	0		
UBA	75	97	167.7	0.56	2	17	2	0		
UBA	92	38	608.9	0.53	2	15	1	0		
UBA	89	13	324.9	0.24	0	11	2	0		
UBA	62	31	867.8	0.09	4	13	2	0		
UBA	97	47	993.4	0.9	2	1	2	0		
UBA	1	100	754.9	0.9	4	17	2	0		
UBA	27	88	99.58	0.63	0	6	2	1		
UBA	62	52	295.8	0.34	4	4	2	0		
UBA	77	56	118	0.35	1	16	2	0		
UBA	3	15	654	0.73	3	15	2	0		

UBA	70	29	767.5	0.9	2	15	2	0		
UBA	72	8	926.4	0.89	3	6	1	0		
UBA	27	5	443.8	0.78	4	15	3	1		
Unity	9	29	902.4	0.64	2	14	2	0		
Unity	62	47	879.1	0.08	4	11	3	0		
Unity	37	68	324	0.16	1	7	2	0		
Unity	97	76	870.2	0.9	0	9	2	1		
Unity	51	45	381.9	0.61	2	16	1	0		
Unity	44	2	303.9	0.01	1	5	1	0		
Unity	24	27	652.6	0.1	1	10	3	0		
Unity	79	95	335.1	0.66	2	18	1	0		
Unity	59	36	633.9	0.01	4	3	1	0		
Unity	32	36	613.7	0.16	1	13	3	0		
Unity	96	26	567.8	0.55	3	11	1	1		
Unity	88	43	386.6	0.69	1	17	2	0		
Unity	52	27	553	0.65	4	13	3	0		
Unity	62	69	600.5	0.22	4	6	3	0		
Unity	58	20	550.2	0.71	3	17	2	0		
Unity	52	11	450	0.24	1	13	1	0		
Unity	12	74	24.52	0.33	4	19	3	0		
Unity	39	38	591.2	0.75	2	19	3	1		
Unity	2	6	176.8	0.65	0	10	2	0		
Unity	3	72	646.2	0.85	0	3	2	0		
Unity	56	23	761.8	0.66	0	14	1	0		
Unity	81	47	505.4	0.57	1	19	3	0		

Unity	59	90	546.9	0.09	3	5	1	0		
Unity	2	46	952	0.37	4	5	1	0		
Unity	2	12	848.7	0.27	2	17	2	0		
Unity	92	90	915.1	0.24	2	10	3	1		
Unity	54	13	815.6	0.97	1	5	3	0		
Unity	87	62	117.8	0.39	0	4	3	0		
Unity	96	82	157.5	0.89	1	6	2	0		
Unity	97	89	531.4	0.63	3	4	2	1		
Unity	1	97	253.6	0.79	3	7	2	0		
Unity	19	60	481.9	0.5	4	7	2	0		
Unity	2	43	400.4	0.58	4	18	2	0		
Unity	53	76	553.9	0.49	3	3	3	0		
Unity	44	100	749.4	0.2	3	1	1	0		
Unity	90	68	718.3	0.72	3	4	3	0		
Unity	32	5	526	0.28	1	6	1	0		
Unity	70	37	824.2	0.02	3	9	2	0		
Unity	32	72	436.6	0.65	2	14	1	0		
Unity	68	92	897.8	0.18	0	12	3	0		
Unity	55	31	80.57	0.94	3	4	3	0		
Unity	75	9	955.1	0.95	0	12	1	0		
Unity	56	51	658.2	0.91	1	10	2	1		
Unity	17	29	386.3	0.37	0	7	2	0		
Unity	38	78	205.2	0.02	1	3	3	1		
Unity	24	40	484.4	0.93	2	14	1	0		
Unity	69	41	202.6	0.43	1	10	2	0		

Unity	98	86	174.7	0.97	1	16	2	1		
Unity	70	11	621.2	0.96	4	16	3	0		
Unity	86	23	527.9	0.85	3	10	2	0		
Unity	11	1	673.4	0.29	3	13	1	0		
Unity	16	46	930.3	0.39	2	6	3	0		
Unity	97	21	536.9	0.85	2	12	3	0		
Unity	73	90	585.1	0.32	3	9	1	0		
Unity	59	36	98.73	0.17	4	13	3	0		
Unity	70	54	720.5	0.56	3	4	2	0		
Unity	80	87	405.9	0.94	4	11	2	0		
Unity	93	57	977	0.7	2	18	1	0		
Unity	3	1	824.9	0.57	0	11	2	0		
Unity	20	63	527.3	0.1	2	4	1	0		
Unity	59	54	132	0.62	2	18	1	1		
Unity	36	55	860.8	0.99	1	1	2	0		
Unity	19	40	801.8	0.14	1	7	1	1		
Unity	90	15	192.7	0.52	1	6	2	0		
Unity	67	21	302.3	0.88	1	15	3	0		
Unity	19	47	795.5	0.74	4	18	3	0		
Unity	20	73	193.4	0.7	3	11	1	0		
Unity	96	53	918.6	0.7	0	15	2	0		
Unity	71	9	430.3	0.36	2	16	2	0		
Unity	52	74	96.17	0.29	4	7	3	0		
Unity	33	52	817.6	0.81	0	10	3	1		
Unity	40	57	379.8	0.81	4	12	2	0		

Unity	39	26	531.2	0.87	4	14	1	0		
Unity	82	41	121.2	0.91	2	5	3	0		
Unity	1	35	12.56	0.51	3	18	1	0		
Unity	11	63	641.6	0.5	1	5	1	0		
Unity	92	25	520	0.8	3	3	3	0		
Unity	57	90	597.6	0.65	2	5	1	1		
Unity	89	75	795.8	0.7	2	17	1	0		
Unity	50	38	650.1	0.8	2	19	2	0		
Unity	23	2	183	0.89	0	19	1	0		
Unity	31	7	367.2	0.34	3	12	2	0		
Unity	94	82	906.8	0.38	3	9	3	0		
Unity	42	91	772.3	0.09	4	1	3	0		
Unity	99	34	561.1	0.58	4	18	2	0		
Unity	7	91	263.1	0.04	4	4	3	0		
Unity	16	17	309	0.47	2	10	1	0		
Unity	90	43	734.6	0.54	3	7	1	0		
Unity	60	59	852.7	0.29	2	1	1	0		
Unity	2	51	674.7	0.59	0	17	3	0		
Unity	1	54	811.4	0.03	4	2	1	0		
Unity	48	24	769	0.04	2	11	3	1		
Unity	12	25	158.2	0.82	2	9	1	1		
Unity	69	71	429.7	0.36	0	7	2	0		
Unity	37	52	161.5	0.13	1	17	3	0		
Unity	32	70	242.9	0.52	3	17	3	0		
Unity	9	88	870.6	0.77	1	2	1	0		

Unity	99	33	478.2	0.22	4	6	3	1		
Unity	19	49	540	0.62	2	14	3	0		
Unity	48	29	186.7	0.09	0	6	2	1		
Wema	80	63	319.2	0.05	0	2	2	1		
Wema	3	22	237.9	0.53	3	1	1	0		
Wema	20	26	214.1	0.54	1	10	2	0		
Wema	24	28	641.6	0.64	3	5	1	0		
Wema	54	85	264.3	0.73	2	2	1	1		
Wema	33	49	635.7	0.98	3	16	1	0		
Wema	24	71	67.87	0.52	0	2	3	1		
Wema	75	81	150.4	0.32	0	10	2	0		
Wema	72	84	281.8	0.8	4	2	1	0		
Wema	36	49	601.1	0.27	4	16	2	1		
Wema	38	20	679.1	0.44	2	16	1	0		
Wema	84	86	448.3	0.08	1	19	2	0		
Wema	99	92	451.3	0.03	3	7	2	0		
Wema	89	63	727.3	0.96	2	12	1	0		
Wema	99	61	25.66	0.84	4	15	3	0		
Wema	25	49	347.7	0.7	3	8	2	0		
Wema	93	71	820.5	0.41	4	6	3	1		
Wema	18	1	607.2	0.17	3	10	3	1		
Wema	82	96	343.9	0.16	1	3	1	0		
Wema	66	13	906.5	0.25	2	19	1	0		
Wema	54	94	774.9	0.55	3	17	1	0		
Wema	35	87	272.3	0.71	0	7	2	0		

Wema	80	51	648.8	0.66	0	4	1	1		
Wema	61	56	599.8	0.28	3	14	1	0		
Wema	41	83	407.2	0.95	3	15	2	0		
Wema	100	62	373.3	0.74	3	8	2	0		
Wema	33	32	350.1	0.55	0	15	1	0		
Wema	68	30	621	0.61	0	14	1	0		
Wema	33	29	448	0.42	3	9	1	0		
Wema	14	49	833.5	0.25	4	8	1	0		
Wema	21	45	672.9	0.36	4	17	3	0		
Wema	48	93	79.23	0.76	1	2	3	0		
Wema	20	30	643.8	0.01	2	3	1	0		
Wema	8	16	278.4	0.12	4	8	3	0		
Wema	7	40	596.9	0.05	0	12	2	0		
Wema	67	19	683.3	0.04	4	1	3	0		
Wema	17	18	238.5	0.86	1	10	3	0		
Wema	33	1	462.3	0.7	1	3	1	0		
Wema	48	78	198.4	0.47	2	6	3	0		
Wema	76	47	61.82	0.1	4	7	3	1		
Wema	59	66	384.7	0.49	0	18	3	1		
Wema	86	92	369.3	0.47	0	11	3	0		
Wema	22	94	995.5	0.17	3	19	2	1		
Wema	30	38	451.9	0.43	0	5	1	0		
Wema	38	51	340.2	0.4	2	19	1	0		
Wema	51	63	584.7	0.62	2	11	2	0		
Wema	54	4	97.88	0.64	0	12	3	0		

Wema	8	1	962.1	0.05	0	10	2	0		
Wema	27	8	638.6	0.37	3	8	3	0		
Wema	27	29	757.3	0.63	0	9	2	0		
Wema	98	55	178.8	0.5	3	4	1	0		
Wema	21	3	415.4	0.86	3	5	1	0		
Wema	30	32	428.5	0.66	0	8	1	1		
Wema	97	10	823.6	0.16	1	9	3	0		
Wema	28	74	11.22	0.07	1	15	3	0		
Wema	64	83	307.8	0.64	3	15	2	0		
Wema	97	34	981.4	0.03	0	16	2	0		
Wema	69	97	480.1	0.59	1	14	1	0		
Wema	61	87	388.7	0.94	0	8	3	0		
Wema	48	55	128.6	0.58	0	18	2	0		
Wema	19	92	974.4	0.39	0	10	3	0		
Wema	4	32	29.83	0.64	4	6	3	0		
Wema	35	50	739.8	0.46	0	12	1	0		
Wema	64	7	770.9	0.55	4	2	1	0		
Wema	49	93	251.2	0.94	3	7	3	0		
Wema	17	8	674.8	0.39	1	16	3	0		
Wema	44	65	102	0.96	1	3	3	0		
Wema	92	57	264.5	0.91	1	13	3	1		
Wema	30	67	790	0.2	1	3	1	1		
Wema	93	88	866	0.07	1	18	2	1		
Wema	46	87	612.4	0.1	1	11	1	0		
Wema	6	59	511.8	0.02	1	8	2	0		

Wema	99	72	167.7	0.09	3	6	3	0		
Wema	37	54	72.48	0.68	0	1	2	0		
Wema	24	67	126.5	0.07	2	1	1	0		
Wema	93	51	235.1	0.32	4	8	1	0		
Wema	46	97	348.3	0.84	2	5	3	1		
Wema	53	92	387.1	0.02	2	16	1	0		
Wema	95	8	352.9	0.81	0	11	3	1		
Wema	99	34	246.7	0.28	0	5	1	0		
Wema	60	35	514.6	0.12	0	4	1	0		
Wema	97	96	827	0.7	1	6	3	0		
Wema	63	88	286.9	0.63	3	14	2	0		
Wema	85	78	943	0.88	1	19	1	0		
Wema	32	32	874.6	0.74	3	9	2	0		
Wema	87	46	170.2	0.8	3	12	1	0		
Wema	33	16	788.4	0.28	2	11	1	0		
Wema	67	68	578.8	0.18	2	14	1	1		
Wema	18	37	570.9	0.75	3	16	1	0		
Wema	25	54	397.1	0.81	3	1	2	1		
Wema	95	85	506.1	0.99	1	13	2	0		
Wema	54	14	403.2	0.41	1	13	1	0		
Wema	58	95	544.3	0.37	2	2	1	0		
Wema	67	55	631.7	0.78	0	4	2	0		
Wema	46	48	677.6	0.34	3	6	3	0		
Wema	24	82	537.2	0.93	1	5	3	0		
Wema	32	7	724.2	0.86	2	3	2	0		

Wema	47	74	149.9	0.43	1	1	1	0		
Wema	86	7	219.9	0.75	3	4	1	0		
Wema	23	33	820.6	0.75	3	2	3	0		
Zenith	66	23	329.3	0.1	4	5	3	0		
Zenith	27	85	32.66	0.9	0	3	2	0		
Zenith	2	19	162.3	0.51	4	9	3	0		
Zenith	90	19	231.4	0.83	2	6	2	0		
Zenith	17	36	957.8	0.32	1	18	3	0		
Zenith	33	29	577.3	0.9	2	6	3	0		
Zenith	9	60	933.4	0.39	3	19	3	0		
Zenith	43	82	579.4	0.01	4	12	1	0		
Zenith	48	2	484.6	0.91	4	11	1	0		
Zenith	39	1	320.6	0.09	4	2	1	0		
Zenith	93	47	610.3	0.32	2	15	3	1		
Zenith	42	69	147.8	0.95	3	12	2	0		
Zenith	26	20	899.3	0.95	1	12	2	0		
Zenith	99	11	831	0.57	1	9	2	0		
Zenith	50	2	324.4	0.63	2	2	2	0		
Zenith	25	67	953.3	0.45	4	3	1	0		
Zenith	24	87	827	0.29	0	12	1	0		
Zenith	13	12	669.6	0.33	2	16	3	0		
Zenith	60	20	130.5	0.67	2	1	3	0		
Zenith	7	5	483.5	0.75	3	3	2	0		
Zenith	57	37	511.1	0.79	1	2	1	0		
Zenith	36	38	240.2	0.79	2	9	2	0		

Zenith	45	94	89.17	0.09	2	19	3	0		
Zenith	20	9	779	0.49	4	15	3	1		
Zenith	65	98	130.8	0.06	4	18	2	0		
Zenith	8	53	598.4	0.55	0	11	3	1		
Zenith	16	44	937.1	0.44	3	16	2	1		
Zenith	14	99	877.3	0.89	0	17	1	0		
Zenith	76	86	285	0.35	4	5	2	0		
Zenith	87	24	852.9	0.12	1	12	1	0		
Zenith	15	81	10.33	0.14	4	11	1	0		
Zenith	92	74	27.51	0.76	3	15	3	0		
Zenith	98	30	606.5	0.62	4	3	3	0		
Zenith	66	59	371.2	0.1	3	1	3	0		
Zenith	32	87	392.6	0.08	3	16	2	0		
Zenith	87	96	568.8	0.7	3	2	2	1		
Zenith	63	81	199.9	0.07	3	2	3	0		
Zenith	86	81	219.6	0.82	3	3	2	0		
Zenith	51	14	489.8	0.71	1	8	3	0		
Zenith	25	9	756.7	0.08	0	15	2	1		
Zenith	58	40	623.3	0.08	2	7	1	0		
Zenith	63	66	304.4	0.99	0	6	1	0		
Zenith	62	25	436	0.37	3	9	3	0		
Zenith	22	73	179	0.37	1	1	2	0		
Zenith	58	22	83.88	0.81	4	8	2	0		
Zenith	58	4	784.9	0.95	1	3	1	1		
Zenith	86	26	272.7	0.99	3	4	1	0		

Zenith	49	58	186	0.75	1	15	3	0		
Zenith	52	29	694.7	0.38	1	6	1	1		
Zenith	42	98	621.9	0.08	3	15	3	0		
Zenith	70	87	636.9	0.78	1	3	1	0		
Zenith	15	37	236.6	0.56	2	14	1	1		
Zenith	54	75	240.3	0.42	2	14	2	0		
Zenith	60	70	228.6	0.91	1	4	1	0		
Zenith	97	18	182	0.11	4	12	1	0		
Zenith	8	85	659.2	0.49	3	17	3	0		
Zenith	53	42	849.5	0.01	2	7	1	0		
Zenith	60	100	727.1	0.47	4	9	3	0		
Zenith	5	41	794.2	0.06	1	3	1	0		
Zenith	68	38	410.8	0.12	4	13	1	1		
Zenith	6	34	657.5	0.12	0	15	3	0		
Zenith	96	17	360.7	0.65	2	18	2	1		
Zenith	94	37	530.3	0.75	1	19	3	0		
Zenith	47	25	486.8	0.58	3	1	3	0		
Zenith	99	76	185.1	0.96	1	6	3	0		
Zenith	55	87	393.2	0.37	3	2	2	0		
Zenith	40	86	964.2	0.29	1	14	1	0		
Zenith	52	85	256.1	0.87	4	6	3	1		
Zenith	16	27	142.5	0.22	4	14	3	0		
Zenith	13	57	424.4	0.96	4	4	2	0		
Zenith	30	31	792.4	0.01	0	8	3	0		
Zenith	19	6	586.3	0.97	3	3	1	0		

Zenith	17	91	684.2	0.04	4	19	2	0		
Zenith	63	40	197.8	0.89	0	14	1	0		
Zenith	19	98	622.8	0.53	3	17	1	0		
Zenith	92	12	330.3	0.99	1	8	2	0		
Zenith	58	53	836	0.07	0	14	3	0		
Zenith	55	71	633	0.55	2	6	1	0		
Zenith	90	10	206.9	0.97	0	16	1	0		
Zenith	90	45	73.97	0.52	0	19	1	1		
Zenith	62	17	52.21	0.63	3	14	1	0		
Zenith	23	26	377.7	0.7	1	19	2	0		
Zenith	9	85	273.2	0.45	2	3	2	0		
Zenith	12	92	649.6	0.63	4	10	1	0		
Zenith	1	62	504.3	0.58	4	19	3	1		
Zenith	58	46	273.4	0.9	0	9	1	0		
Zenith	1	64	157.8	0.05	3	13	3	0		
Zenith	34	2	135.9	0.28	3	7	1	0		
Zenith	96	54	866	0.95	4	14	3	0		
Zenith	48	65	251	0.89	2	9	1	0		
Zenith	89	51	655.2	0.46	1	19	3	0		
Zenith	1	53	812.6	0.62	2	15	1	0		
Zenith	16	36	617.9	0.28	1	14	1	0		
Zenith	61	26	861	0.19	4	6	3	0		
Zenith	64	29	398.5	0.46	1	4	2	1		
Zenith	63	21	956.7	0.35	3	8	3	0		
Zenith	69	11	141.8	0.58	0	17	2	0		

Zenith	22	40	111.1	0.08	0	9	1	0		
Zenith	93	11	662.1	0.97	4	1	3	1		
Zenith	67	36	943.4	0.99	3	14	3	0		
Access	76	59	810.8	0.7	2	11	3	1		
Access	26	39	85.65	0.54	3	3	2	0		
Access	16	99	286.6	0.31	0	9	2	0		
Access	51	54	931	0.81	0	14	1	0		
Access	86	98	260.5	0.68	3	9	2	0		
Access	57	55	783.4	0.16	3	4	2	0		
Access	29	24	826.9	0.91	0	4	1	0		
Access	78	22	500.5	0.82	2	8	3	0		
Access	92	69	392.5	0.95	4	2	3	0		
Access	69	56	455.6	0.73	4	9	1	0		
Access	47	33	234.7	0.61	3	14	3	0		
Access	94	36	890.5	0.42	4	9	2	0		
Access	62	20	456.1	0.93	0	16	1	0		
Access	69	83	883.6	0.87	0	17	1	0		
Access	76	99	114.6	0.05	3	16	1	0		
Access	16	81	988.4	0.03	3	12	3	1		
Access	90	62	448.8	0.38	3	3	1	0		
Access	90	98	57.35	0.81	2	11	1	0		
Access	48	73	47.64	0.99	4	14	1	0		
Access	85	26	748.8	0.15	3	18	3	1		
Access	39	66	662.5	0.59	1	5	2	0		
Access	100	73	804.1	0.38	1	16	1	0		

Access	33	73	114.3	0.97	0	5	2	1		
Access	94	40	45.22	0.84	4	3	3	0		
Access	23	17	513.5	0.84	3	12	3	0		
Access	10	1	393.7	0.47	1	1	3	0		
Access	69	89	467.3	0.41	4	9	3	0		
Access	100	61	991.7	0.27	0	9	2	0		
Access	34	43	971	0.06	2	4	1	0		
Access	52	42	942.9	0.86	4	3	3	0		
Access	95	25	786	0.81	4	17	3	1		
Access	10	39	361.3	1	2	2	2	0		
Access	19	35	418	1	4	1	2	0		
Access	58	3	554.1	0.56	0	8	3	0		
Access	96	44	832.2	0.77	0	13	1	0		
Access	1	51	78.03	0.94	2	15	1	0		
Access	69	94	430.1	0.85	3	8	3	0		
Access	4	98	91.79	0.25	1	12	2	0		
Access	16	12	519.5	0.45	3	12	3	0		
Access	24	19	899.9	0.13	4	14	3	0		
Access	80	44	784.7	0.95	1	15	2	0		
Access	2	59	235.6	0.61	2	15	3	0		
Access	92	49	976.2	0.23	2	9	3	0		
Access	32	61	670.7	0.67	1	19	3	0		
Access	91	17	20.32	0.62	0	8	2	1		
Access	84	74	262.9	0.36	1	19	1	0		
Access	24	57	268.1	0.11	4	6	1	0		

Access	12	55	78.56	0.67	0	19	2	0		
Access	50	47	960.2	0.52	2	4	1	0		
Access	35	12	483.5	0.77	1	7	3	0		
Access	33	62	896.9	0.52	3	4	1	1		
Access	33	80	329	0.85	3	12	3	0		
Access	61	88	134	0.55	4	13	1	0		
Access	51	83	479.6	0.56	3	3	2	0		
Access	43	8	122.8	0.88	1	12	1	0		
Access	12	95	491.2	0.4	1	2	1	0		
Access	67	21	921.1	0.13	3	2	1	0		
Access	65	81	525.6	0.03	1	9	2	0		
Access	33	87	620.3	0.76	4	1	1	0		
Access	40	80	165.1	0.62	0	8	2	0		
Access	74	70	53.3	0.7	4	6	3	0		
Access	43	72	119.7	0.21	4	6	1	1		
Access	44	25	602.6	0.14	1	18	2	0		
Access	29	82	104.1	0.01	0	17	2	0		
Access	13	89	382.2	0.35	1	12	1	0		
Access	12	12	554.6	0.59	2	6	3	1		
Access	95	15	394.8	0.39	1	8	2	1		
Access	46	59	172.1	0.44	1	19	1	0		
Access	2	26	281.1	0.9	1	10	3	0		
Access	35	26	929	0.35	0	1	2	0		
Access	87	47	679.2	0.51	1	13	2	0		
Access	81	32	501.1	0.78	0	17	3	0		

Access	90	10	22.59	0.4	3	3	3	0		
Access	8	16	992.5	0.62	2	8	1	0		
Access	93	71	674.7	0.86	0	14	1	0		
Access	26	17	323.8	0.95	3	3	2	0		
Access	74	23	880.5	0.15	0	16	3	1		
Access	90	26	265.9	0.93	0	13	3	0		
Access	34	85	548.6	0.49	2	17	3	0		
Access	7	86	639.7	0.26	4	13	2	0		
Access	68	7	228.9	0.46	4	6	3	0		
Access	58	14	978.6	0.98	0	17	3	0		
Access	75	79	750.7	0.49	1	14	2	0		
Access	29	7	562.2	0.33	3	17	1	1		
Access	36	9	188.8	0.63	1	8	1	0		
Access	89	48	63.32	0.24	1	11	2	0		
Access	21	72	326.9	0.08	0	1	3	0		
Access	36	59	755.7	0.13	0	7	2	0		
Access	10	87	276.6	0.13	0	17	1	0		
Access	73	93	544.2	0.15	0	10	1	0		
Access	24	82	730.7	0.14	4	11	2	0		
Access	64	95	116.9	0.64	0	9	2	0		
Access	99	94	688.5	0.18	4	9	1	0		
Access	49	39	101	0.35	4	5	2	0		
Access	99	99	965.5	0.9	0	17	1	0		
Access	36	18	493.5	0.47	2	9	2	0		
Access	82	59	159.2	0.67	3	17	2	0		

Access	96	17	349	0.17	1	19	2	0		
Access	24	14	419.3	0.19	4	7	1	1		
Access	23	31	760.6	0.04	3	9	2	0		
FBN	62	24	746	0.17	2	7	1	0		
FBN	96	99	615.8	0.28	1	13	1	1		
FBN	37	60	114.9	0.18	1	11	2	0		
FBN	12	45	160.1	0.09	4	14	2	0		
FBN	55	98	102.8	0.12	0	19	1	0		
FBN	13	3	581.4	0.46	2	11	1	0		
FBN	23	37	878.4	0.21	4	13	3	0		
FBN	89	43	439.5	0.36	2	14	1	0		
FBN	99	40	169.2	0.5	0	13	3	0		
FBN	30	90	562.5	0.69	0	1	3	0		
FBN	17	55	813.6	0.04	4	8	3	1		
FBN	62	87	744.3	0.8	0	12	2	0		
FBN	84	39	882.6	0.63	3	19	2	1		
FBN	89	15	285.1	0.08	1	12	1	0		
FBN	86	4	202	0.87	4	8	3	0		
FBN	13	93	881	0.92	2	8	2	0		
FBN	59	86	661	0.06	3	13	2	0		
FBN	19	25	317.8	0.28	4	12	2	0		
FBN	49	13	744.9	0.81	0	16	2	0		
FBN	100	82	817	0.75	0	1	2	0		
FBN	12	33	618.8	0.18	0	13	3	1		
FBN	61	16	691	0.21	0	7	3	0		

FBN	19	42	381.5	0.37	2	5	3	0		
FBN	76	66	543.3	0.48	3	8	3	0		
FBN	9	55	528.9	0.62	1	14	1	0		
FBN	71	99	849.6	0.37	2	13	3	0		
FBN	28	42	841.6	0.46	2	15	2	0		
FBN	78	34	834.4	0.75	1	17	3	0		
FBN	95	30	20.92	0.04	1	15	1	1		
FBN	52	13	752.1	0.25	0	18	2	0		
FBN	83	13	761.1	0.71	2	18	3	0		
FBN	16	18	681.6	0.9	4	2	3	0		
FBN	69	32	758.7	0.51	1	4	3	0		
FBN	99	96	843.6	0.53	1	1	2	0		
FBN	12	99	475.6	0.11	3	4	2	0		
FBN	25	39	507	0.45	4	18	1	1		
FBN	52	46	133.5	0.53	4	5	2	0		
FBN	85	29	391.5	0.24	1	19	2	0		
FBN	100	62	646.9	0.27	0	15	3	1		
FBN	53	93	782.6	0.38	3	8	2	0		
FBN	23	62	225	0.02	4	17	3	0		
FBN	16	57	146.4	0.32	1	1	3	0		
FBN	57	16	666.4	0.21	4	12	2	0		
FBN	39	56	11.86	0.33	3	19	1	0		
FBN	53	10	186.6	0.12	3	15	1	0		
FBN	42	30	38.78	0.89	2	9	2	0		
FBN	58	25	175.2	0.59	1	7	1	0		

FBN	39	84	189.5	0.68	3	2	3	0		
FBN	14	5	393.8	0.79	2	9	3	0		
FBN	95	65	408.3	0.5	2	10	2	0		
FBN	5	94	875.5	0.09	4	2	1	0		
FBN	35	49	515	0.54	4	19	1	0		
FBN	87	3	726.2	0.59	2	11	3	0		
FBN	93	45	113.1	0.75	4	13	2	0		
FBN	75	14	233.2	0.43	3	10	1	1		
FBN	18	30	705.9	0.13	0	14	2	0		
FBN	76	68	62.28	0.28	3	18	1	0		
FBN	9	18	491.7	0.36	2	19	3	0		
FBN	74	62	603.4	0.65	2	9	3	0		
FBN	58	37	606.1	0.57	3	7	2	0		
FBN	17	25	32.7	0.36	4	3	3	1		
FBN	7	48	328.9	0.99	4	6	1	0		
FBN	46	65	640.1	0.61	2	2	3	0		
FBN	13	53	157.5	0.24	1	6	2	0		
FBN	40	79	698.5	0.1	4	3	2	1		
FBN	42	73	748.5	0.15	3	16	2	0		
FBN	9	15	982.2	0.25	4	8	1	1		
FBN	50	49	217.7	0.16	3	2	1	0		
FBN	27	88	826.9	0.19	2	17	1	0		
FBN	66	68	525.1	0.29	4	9	1	0		
FBN	5	12	379.7	0.17	3	12	1	0		
FBN	29	59	131.2	0.9	2	3	3	0		

FBN	37	37	738.2	0.08	1	11	1	0		
FBN	38	61	243.5	0.52	3	1	1	0		
FBN	83	43	377.7	0.41	1	13	3	0		
FBN	8	70	71.92	0.98	1	15	1	0		
FBN	65	39	803.4	0.11	0	5	2	0		
FBN	86	56	768.9	0.4	4	17	1	0		
FBN	17	63	736.3	0.97	3	6	1	0		
FBN	71	46	306.5	0.87	1	11	2	0		
FBN	89	88	181.8	0.82	0	18	2	0		
FBN	45	11	641.2	0.26	2	4	2	0		
FBN	4	62	501.2	0.17	3	3	2	0		
FBN	36	77	186.5	0.67	1	13	1	0		
FBN	70	98	689.1	0.93	4	9	2	0		
FBN	31	25	991.2	0.56	0	6	2	0		
FBN	19	71	749.8	0.57	1	12	3	0		
FBN	61	52	742.6	0.28	4	2	2	0		
FBN	54	4	950.8	0.77	1	3	2	0		
FBN	39	59	210.9	0.19	3	3	3	0		
FBN	91	72	569.9	0.32	3	12	3	0		
FBN	74	20	979.9	0.43	3	15	1	0		
FBN	90	93	173.6	0.51	0	9	3	0		
FBN	19	63	703.3	0.24	3	8	1	0		
FBN	39	54	590	0.11	4	11	3	0		
FBN	67	74	590.6	0.61	1	16	3	0		
FBN	45	98	700.7	0.29	2	4	1	1		

FBN	13	57	930.3	0.58	4	1	3	0		
FBN	92	90	784.8	0.15	4	7	1	0		
FBN	58	41	65.57	0.48	3	2	3	0		
Union	20	3	854.2	0.53	0	5	2	0		
Union	92	6	233.3	0.05	1	7	3	0		
Union	72	5	383.2	0.34	0	4	1	0		
Union	61	5	467.1	0.13	1	5	3	0		
Union	39	54	486.1	0.06	1	1	1	0		
Union	1	47	440.8	0.99	0	1	1	1		
Union	3	87	938	0.32	4	15	2	0		
Union	77	49	709	0.81	1	19	2	0		
Union	92	9	634.9	0.25	0	10	1	0		
Union	62	99	59.34	0.68	1	18	3	0		
Union	63	20	904.9	0.76	2	3	1	0		
Union	25	61	910.3	0.6	4	11	1	0		
Union	56	35	782.4	0.47	3	15	3	0		
Union	33	50	16.96	0.41	1	11	3	0		
Union	38	82	66.71	0.35	2	17	3	0		
Union	6	62	233	0.93	3	17	3	0		
Union	58	17	140	0.83	3	6	1	0		
Union	44	88	404.9	0.97	1	5	1	1		
Union	45	3	923.4	0.12	0	8	2	0		
Union	32	32	477.3	0.73	1	2	1	0		
Union	45	99	23.27	0.94	2	16	3	0		
Union	61	13	289.9	0.18	3	12	2	0		

Union	47	89	54.15	0.07	1	6	3	0		
Union	21	73	45.92	0.74	0	1	1	0		
Union	80	68	283	0.57	1	4	2	1		
Union	85	14	282.9	0.84	4	17	3	0		
Union	75	98	789.3	0.14	0	13	2	0		
Union	36	11	975.1	0.8	2	18	2	0		
Union	99	56	377.5	0.2	3	5	1	0		
Union	19	67	434.6	0.16	1	3	3	0		
Union	20	92	846.3	0.16	0	11	2	0		
Union	57	29	275.5	0.81	4	13	2	0		
Union	18	9	164.8	0.67	0	4	3	0		
Union	47	67	854.6	0.52	4	14	2	0		
Union	49	25	56.04	0.36	2	11	2	1		
Union	14	98	623.5	0.88	3	10	2	0		
Union	15	76	566.4	0.39	2	19	3	0		
Union	31	42	352.5	0.82	1	18	3	0		
Union	1	9	322	0.44	0	6	3	0		
Union	54	79	774.1	0.38	4	15	3	0		
Union	3	40	95.56	0.46	4	10	1	0		
Union	16	25	244.4	0.3	0	5	2	0		
Union	87	5	735.1	0.75	0	5	3	0		
Union	57	11	552.8	0.5	1	10	2	0		
Union	75	59	136.7	0.23	2	5	2	1		
Union	12	98	883.3	0.9	4	3	1	0		
Union	74	88	399.5	0.38	1	6	3	0		

Union	96	79	829.6	0.54	2	4	3	0		
Union	16	38	789.8	0.91	1	18	2	0		
Union	72	72	628.2	0.62	4	9	3	0		
Union	76	5	499.5	0.12	0	17	2	0		
Union	24	48	979.8	0.94	0	13	2	0		
Union	28	90	153.5	0.63	2	15	1	0		
Union	8	18	727.9	0.33	3	17	1	0		
Union	92	70	199.3	0.14	2	13	1	0		
Union	36	37	344.9	0.79	3	16	2	0		
Union	90	60	315.6	0.62	1	4	3	0		
Union	8	48	947.6	0.53	1	5	1	0		
Union	58	77	967.7	0.89	4	5	3	0		
Union	60	28	191.2	0.79	3	9	1	0		
Union	50	69	31.81	0.15	4	17	1	0		
Union	28	77	722.8	0.31	0	13	2	0		
Union	92	81	423.8	0.25	3	2	1	0		
Union	41	40	97.95	0.74	2	13	1	0		
Union	100	45	686.9	0.03	0	19	2	0		
Union	64	96	549.7	0.57	0	3	3	0		
Union	27	62	980.7	0.76	2	10	1	0		
Union	63	93	235.1	0.88	0	2	2	0		
Union	17	58	409.9	0.34	3	13	1	0		
Union	73	67	547.1	0.82	4	19	2	0		
Union	33	56	793.8	0.11	0	17	3	0		
Union	84	40	553.4	0.85	3	14	3	1		

Union	77	98	616.8	0.13	1	12	2	0		
Union	92	40	233.9	0.4	3	12	2	0		
Union	29	91	848.5	0.8	4	11	2	1		
Union	13	77	434.3	0.15	3	7	3	1		
Union	46	33	149.6	0.23	2	4	1	0		
Union	35	6	667.6	0.72	4	19	3	0		
Union	6	88	223.9	0.72	1	1	2	0		
Union	82	19	106.6	0.64	2	3	2	0		
Union	69	80	863.4	0.69	3	13	3	0		
Union	47	40	465.6	0.54	4	15	2	0		
Union	25	92	751.8	0.25	1	2	3	0		
Union	66	97	297.7	0.35	2	11	2	1		
Union	10	59	789.1	0.18	1	4	1	0		
Union	56	59	120.8	0.91	3	19	1	0		
Union	30	48	362.4	0.58	1	17	2	0		
Union	5	41	978.3	0.4	0	3	3	0		
Union	33	13	796.1	0.46	2	18	1	0		
Union	65	45	502.3	0.95	1	8	3	0		
Union	18	89	346.4	0.15	3	19	2	0		
Union	96	53	873.4	0.59	3	5	3	0		
Union	49	42	549.4	0.51	1	7	1	0		
Union	11	100	760.8	0.61	3	8	3	1		
Union	85	63	756.6	0.02	4	5	3	0		
Union	26	27	27.77	0.87	4	17	2	0		
Union	63	38	135	0.93	4	5	2	0		

Union	89	60	663	0.57	0	4	2	0		
Union	86	7	170	0.7	1	19	1	0		
Union	59	5	262.9	0.92	0	2	1	0		
Polaris	27	45	947.2	0.71	3	11	2	0		
Polaris	49	58	564.6	0.15	0	13	2	0		
Polaris	77	31	753.2	0.58	3	15	1	0		
Polaris	33	23	562.4	0.61	3	12	1	0		
Polaris	98	42	557	0.42	2	11	1	0		
Polaris	99	10	103.2	0.74	4	7	1	0		
Polaris	1	32	350.2	0.93	3	5	2	0		
Polaris	21	53	289.6	0.93	2	17	1	0		
Polaris	55	30	396.8	0.45	4	9	2	0		
Polaris	6	39	800.4	0.11	3	14	3	0		
Polaris	92	31	378.2	0.98	3	12	3	0		
Polaris	81	82	457.2	0.84	0	6	1	0		
Polaris	69	100	403.4	0.12	0	17	3	0		
Polaris	95	2	102.5	0.92	4	19	3	0		
Polaris	5	78	446.7	0.87	3	19	3	0		
Polaris	3	77	203.7	0.52	1	16	1	0		
Polaris	53	93	194.9	0.59	0	12	1	0		
Polaris	23	12	103.1	0.4	4	12	3	0		
Polaris	53	39	363.1	0.05	3	6	2	0		
Polaris	37	66	759.9	0.34	3	4	3	1		
Polaris	74	96	283.7	0.8	1	6	1	0		
Polaris	74	13	319.2	0	4	11	1	1		

Polaris	83	46	646.3	0.33	3	1	1	1		
Polaris	17	27	203.9	0.4	0	15	3	0		
Polaris	85	17	898	0.54	1	11	2	0		
Polaris	78	95	525	0.92	4	19	1	1		
Polaris	73	98	453.1	0.35	1	18	1	0		
Polaris	1	60	562.2	0.35	0	17	3	0		
Polaris	51	64	174.4	0.74	4	19	2	0		
Polaris	45	88	617.3	0.45	1	14	3	0		
Polaris	77	35	369.5	0.22	2	18	1	0		
Polaris	4	70	898.2	0.45	0	19	2	0		
Polaris	62	83	177.2	0.14	0	15	2	0		
Polaris	65	30	512	0.18	0	15	1	0		
Polaris	32	71	602.1	0.5	2	4	2	1		
Polaris	34	74	43.05	0.42	2	7	2	0		
Polaris	92	96	370.1	0.91	3	13	3	0		
Polaris	95	98	735.3	0.36	1	3	1	0		
Polaris	72	63	822.2	0.58	3	16	1	0		
Polaris	39	28	320	0.63	1	9	1	0		
Polaris	26	60	800.6	0.01	2	8	1	1		
Polaris	34	79	968.1	0.66	2	13	1	0		
Polaris	54	94	858.3	0.18	3	17	2	0		
Polaris	3	37	126.5	0.96	4	17	3	0		
Polaris	50	93	742.3	0.15	3	17	1	0		
Polaris	12	22	326.5	0.41	1	1	3	1		
Polaris	65	68	988.5	0.09	3	8	3	0		

Polaris	54	20	490.5	1	0	19	2	0		
Polaris	5	96	131.2	0.5	1	6	1	0		
Polaris	94	100	727.9	0.6	1	3	2	1		
Polaris	94	68	329.8	0.07	4	5	1	0		
Polaris	57	91	997.1	0.75	2	13	2	0		
Polaris	17	64	980	0.21	1	15	3	0		
Polaris	47	17	659.6	0.9	2	14	1	1		
Polaris	23	12	138.5	0.21	3	16	3	0		
Polaris	79	22	690	0.19	0	18	3	0		
Polaris	85	17	102.1	0.04	4	11	2	0		
Polaris	14	10	39.07	0.47	2	3	1	0		
Polaris	66	65	824.7	0.56	1	10	3	0		
Polaris	75	22	217.7	0.07	2	10	2	0		
Polaris	51	90	385.2	0.78	1	16	1	1		
Polaris	38	82	973.7	0.45	2	12	1	0		
Polaris	64	50	343.5	0.52	1	2	3	1		
Polaris	98	14	303.5	0.44	3	9	1	0		
Polaris	38	64	919	0.4	1	3	3	0		
Polaris	50	47	119.6	0.56	1	15	1	0		
Polaris	98	29	762	0.16	0	3	2	0		
Polaris	82	76	988.2	0.18	1	19	3	1		
Polaris	30	36	902.3	0.86	0	18	3	0		
Polaris	79	99	655.2	0.95	4	8	1	0		
Polaris	91	52	320.9	0.37	0	17	1	0		
Polaris	51	76	261.8	0.27	3	8	3	0		

Polaris	63	7	821.1	0.64	0	7	1	0		
Polaris	98	29	881	0.41	4	7	2	0		
Polaris	52	11	530.2	0.03	3	3	1	1		
Polaris	38	33	251.9	0.16	2	1	2	0		
Polaris	97	6	121.8	0.72	2	4	2	0		
Polaris	88	79	96.31	0.66	0	5	2	0		
Polaris	79	32	721.9	0.03	0	1	2	1		
Polaris	30	51	425.9	0.22	4	15	2	0		
Polaris	51	75	906.3	0.23	4	5	1	0		
Polaris	81	58	742.3	0.67	1	13	2	0		
Polaris	5	87	199.4	0.02	4	8	1	0		
Polaris	29	26	617.6	0.1	1	8	2	0		
Polaris	4	98	441.2	0.8	3	5	3	1		
Polaris	10	46	581.5	0.18	0	9	2	0		
Polaris	56	98	740.7	0.65	3	1	3	0		
Polaris	17	29	808.1	0.24	4	4	2	0		
Polaris	74	4	845.3	0.1	3	3	2	0		
Polaris	17	81	261.4	0.24	3	2	2	0		
Polaris	84	99	40.8	0.72	3	8	3	0		
Polaris	88	13	741.9	0.86	4	9	3	0		
Polaris	69	39	115.3	0.83	3	2	2	0		
Polaris	34	15	321.8	0.4	1	14	1	0		
Polaris	6	29	13.78	0.67	4	14	2	0		
Polaris	53	29	684.9	0.2	3	4	3	1		
Polaris	66	75	175.5	0.29	4	6	2	0		

Polaris	77	32	658.7	0.9	2	11	3	0		
Polaris	43	2	991.8	0.01	1	3	3	0		
Polaris	75	47	81.94	0.09	3	8	3	0		
Keystone	23	78	86.31	0.21	2	3	2	0		
Keystone	55	84	347	0.03	2	14	3	0		
Keystone	80	22	867.7	0.18	4	13	3	0		
Keystone	95	75	223.4	0.58	2	13	2	0		
Keystone	75	27	366.8	0.42	1	2	3	0		
Keystone	16	67	77.82	0.89	1	13	2	0		
Keystone	8	81	765.5	0.82	0	19	3	0		
Keystone	4	56	414.2	0.34	1	6	1	0		
Keystone	4	92	761.7	0.26	2	16	1	0		
Keystone	56	93	904.6	0.38	1	2	3	0		
Keystone	25	21	240.4	0.59	4	8	1	0		
Keystone	67	47	809.3	0.27	0	17	2	0		
Keystone	96	80	68.5	0.62	2	2	1	0		
Keystone	67	59	899.9	0.41	0	10	3	1		
Keystone	27	84	654.3	0.55	2	16	3	0		
Keystone	93	6	875.5	0.44	4	6	2	0		
Keystone	32	19	10.19	0.29	0	2	3	0		
Keystone	50	57	177.4	0.95	3	2	2	0		
Keystone	61	69	395.9	0.76	2	13	1	1		
Keystone	51	3	385.7	0.14	3	15	1	0		
Keystone	19	1	234.5	0.87	0	2	2	0		
Keystone	21	58	387.8	0.49	3	7	2	0		

Keystone	5	38	137.6	0.89	3	3	3	0		
Keystone	82	15	418.6	0.8	0	8	1	0		
Keystone	92	34	804.3	0.43	4	14	2	0		
Keystone	42	61	484.3	0.02	3	7	1	1		
Keystone	61	35	27.82	0.27	3	7	2	0		
Keystone	22	64	427.2	0.54	4	1	1	0		
Keystone	21	25	36.9	0.63	3	17	3	1		
Keystone	70	24	966.6	0.26	3	9	1	0		
Keystone	1	12	136.2	0.14	3	13	1	1		
Keystone	5	18	710.6	0.83	1	19	1	1		
Keystone	12	15	973.1	0.98	4	19	1	0		
Keystone	90	80	827.8	0.53	3	17	1	0		
Keystone	46	27	702.3	0.17	4	10	3	0		
Keystone	34	72	841.1	0.27	2	10	2	1		
Keystone	49	60	982	0.02	2	4	1	0		
Keystone	78	46	625.1	0.91	3	8	3	0		
Keystone	90	89	563.6	0.12	4	18	1	1		
Keystone	45	17	632	0.58	4	14	1	1		
Keystone	27	93	537.4	0.27	4	18	2	0		
Keystone	73	53	638.7	0.55	1	14	1	0		
Keystone	26	8	455.3	0.65	2	15	3	1		
Keystone	47	79	226.3	0.83	4	4	3	0		
Keystone	86	27	129	0.21	4	13	2	1		
Keystone	56	46	77.66	0.01	1	8	1	0		
Keystone	94	50	270.7	0.14	1	18	3	0		

Keystone	63	71	55.66	0.9	4	12	1	0		
Keystone	48	3	571.5	0.87	0	14	1	0		
Keystone	61	93	291	0.6	4	2	1	0		
Keystone	81	64	826.3	0.6	4	18	2	0		
Keystone	26	44	83.5	0.67	1	2	1	0		
Keystone	36	53	234.1	0.18	1	11	3	0		
Keystone	1	24	732.9	0.91	4	9	1	0		
Keystone	8	60	602.8	0.42	4	12	3	0		
Keystone	99	3	73.85	0.38	4	17	1	0		
Keystone	52	20	878.5	0.52	1	15	3	0		
Keystone	79	46	878.5	0.05	4	17	3	0		
Keystone	47	15	337.1	0.17	4	18	1	0		
Keystone	56	25	347	0.74	3	6	1	0		
Keystone	86	94	989.1	0.08	0	17	3	0		
Keystone	14	78	629.9	0.6	2	18	2	1		
Keystone	90	71	536	0.25	0	8	1	1		
Keystone	28	17	839.8	0.39	3	9	2	0		
Keystone	87	84	292.4	0.29	2	6	1	1		
Keystone	78	100	434.9	0.36	4	5	1	0		
Keystone	88	9	275.1	0.72	3	12	1	0		
Keystone	2	38	791.3	0.3	4	11	1	0		
Keystone	26	48	577.6	0.57	1	1	2	0		
Keystone	14	60	662.2	0.48	3	11	1	0		
Keystone	59	63	259.8	0.66	1	19	3	0		
Keystone	56	86	228	0.94	3	9	3	0		

Keystone	7	2	493.5	0.73	3	2	2	0		
Keystone	3	88	740.2	0.21	1	12	1	0		
Keystone	23	72	530	0.03	2	4	3	0		
Keystone	18	11	42.32	0.26	0	4	2	0		
Keystone	38	64	869.9	0.6	4	18	1	0		
Keystone	99	77	74.04	0.05	3	15	3	0		
Keystone	15	81	956.3	0.5	4	19	3	0		
Keystone	64	72	34.89	0.6	1	13	2	1		
Keystone	89	21	697.5	0.33	4	19	1	0		
Keystone	28	18	91.38	0.77	1	17	1	0		
Keystone	74	94	911.6	0.11	4	6	3	0		
Keystone	39	64	469.8	0.08	0	16	3	0		
Keystone	57	82	146.2	0.73	0	12	3	0		
Keystone	17	95	347.4	0.5	0	19	1	0		
Keystone	86	38	173.2	0.69	0	5	2	0		
Keystone	90	32	178.9	0.43	0	17	2	0		
Keystone	44	11	948.6	0.25	0	1	1	0		
Keystone	25	45	450.8	0.82	1	10	2	0		
Keystone	17	89	790.9	0.8	3	15	3	0		
Keystone	13	33	643.2	0.69	4	11	2	0		
Keystone	84	41	679.7	0.27	0	13	2	0		
Keystone	25	8	140.4	0.59	1	5	2	0		
Keystone	68	11	241.8	0.36	1	1	3	0		
Keystone	10	86	788.9	0.09	1	12	1	0		
Keystone	67	51	971.1	0.92	3	11	3	0		

Keystone	18	88	270.1	0.14	4	17	1	0		
Keystone	100	41	278.9	0.95	3	4	1	1		
Keystone	86	17	478.2	0.45	2	15	1	0		