

Developing novel theorems in graph theory for computational algorithm enhancement

Tanveer Ahmad Wani[@]

*Department of Physics
Noida International University
Noida 203201
Uttar Pradesh
Delhi*

Pradnya Borkar[†]

*Department of Computer Science and Engineering
Symbiosis Institute of Technology
Nagpur Campus
Symbiosis International (Deemed University)
Pune 440027
Maharashtra
India*

Rupali Suraskar[§]

*L&T Edutech
Nagpur 440002
Maharashtra
India*

J. Jabez[‡]

*Department of Information Technology
Sathyabama Institute of Science and Technology
Chennai 600119
Tamil Nadu
India*

[@] E-mail: tanveer.ahmad@niu.edu.in

[†] E-mail: pradnyaborkar2@gmail.com

[§] E-mail: rupali.jumbadkar@gmail.com

[‡] E-mail: jabez.it@sathyabama.ac.in

V. Mamatha *

Department of Computer Science and Information Technology
ARKA JAIN University
Jamshedpur 831001
Jharkhand
India

Divya Sharma #

Centre of Research Impact and Outcome
Chitkara University
Rajpura 140417
Punjab
India

Abstract

Graph theory has been a long time tool for the basis of efficient algorithms in many computational domains such as routing, social network analysis and optimization. Traditional methods, however, tend to process dense and complex graph structures and, therefore, add to the computational cost. The existing approaches rarely involve entropy-based insights to selectively collapse graphs without losing essential structural information. Filling in this gap, the current study proposes a new theoretical construct, i.e., the Graph Edge Entropy Theorem (GEET). The theorem assigns to each edge an amount of informational contribution, quantified using the entropy measures formed from the normalized weights, and allowing the pruning of the low-entropy edges before the execution of the algorithm. To authenticate the efficiency of GEET, the Bitcoin Alpha trust network dataset was used. Preprocessing steps taken were edge weight normalization, and conversion into undirected graphs through mutual trust aggregation. GEET was plugged into Dijkstra's algorithm with the results being benchmarked against traditional and partially pruned graphs. Experimental results showed that GEET enhanced model had significantly reduced execution time and memory usage with high path accuracy. To be precise, it delivered 61% faster computation and reduced memory consumption by 47% as compared to the baseline with a small 0.8% in path accuracy deviation. These results support the applicability of the practical efficiency of entropy-based edge evaluation in the computational graph algorithms.

Subject Classification: Primary 93A30, Secondary 49K15.

Keywords: Graph theory, Entropy-based pruning, Computational efficiency, Dijkstra's algorithm.

1. Introduction

Graph theory is a building block in designing and analyzing computational algorithms so that it is possible to effectively model complicated interrelations

* E-mail: mamatha.v@arkajainuniversity.ac.in (Corresponding Author)

E-mail: divya.sharma.orp@chitkara.edu.in

in a variety of areas like network routing, data mining, cyber security, and social networks.[1] [2] The modeling of the real-world systems as graphs of nodes and edges allows following a structured way to solve many problems related to connectivity, path optimization, influence propagation and resource allocation. The classical algorithms which include Dijkstra, Kruskal and Prim's algorithm have provided the foundation for the solution of shortest path and minimum spanning tree problem. However, as the complexity and the size of the datasets grow, the performance limitations of the traditional methods become apparent[3] [4].

In modern usages, large networks tend to possess redundancy, sparsity, or unequal importance of edges. All existing computational methods consider each edge in the graphs as equally important, which results in an overprocess of meaningless connections [5][6]. This leads to longer execution times and memory usage, and this is crucial in time-sensitive or resource limited contexts. In addition, traditional approaches to simplification of graphs, for example, random edge sampling or degree-based filtering, can degrade the validity of essential structural properties, thus, undermining the precision of the algorithms they facilitate [6][7].

In order to overcome the drawbacks, the current research provides a new entropy-based method to measure the information content of edges in a graph. The Graph Edge Entropy Theorem (GEET) is suggested as a mathematical way of defining and calculating the contribution of each edge for a given weight distribution. Edges with low entropy i.e. lesser informational gain are selectively eliminated before implementing computational algorithms. This pruning strategy targets to preserve the most informational structural paths while optimizing the computational complexity [8]. Unlike the existing methods based on degree centrality or heuristic thresholds, GEET provides a mathematically sound, scalable way of optimizing graph-based computations [9].

The methodology is assessed using the Bitcoin Alpha Trust Network data set that contains directed, weighted and signed trust relationships. Preprocessing involves normalization of edge weights and conversion to an undirected form by aggregating mutual trust scores. Integrating GEET with Dijkstra's algorithm, this study quantifies the improvements in execution time, memory usage as well as path accuracy. The main purposes of this study are:

- To state and verify the Graph Edge Entropy Theorem (GEET) as a new theoretical instrument in graph theory to quantify the edge informativeness for computational improvement.
- To incorporate GEET into a classical graph algorithm and empirically determine the effect of GEET on computational efficiency and path accuracy with the help of a real dataset.

With these objectives, the research should contribute to the advancement of the theoretical basis and practical use of techniques of graph simplification in the direction of more effective and intelligent algorithmic design.

2. Dataset Description and Preprocessing

2.1. Dataset

The Bitcoin Alpha dataset is a directed, signed, and weighted social trust network, where nodes stand for users and edges represent trust scores, from -10 (complete distrust) to +10 (complete trust). The dataset, sourced from the Bitcoin Alpha trading platform, shows how people rate each other's reliability. It has about 3783 nodes and 24186 edges. The distribution of edge weights is skewed, and the positive scores are more common, indicating overall trust-centric interactions. This dataset is a detailed framework for measuring graph-based computational algorithms in a real-world social setting.

2.2. Preprocessing methods implemented

Preprocessing is a very important step in preparing data for graph for algorithmic analysis. For the Bitcoin Alpha dataset, that includes weighted, directed trust relationships, raw data should be transformed into a consistent and simplified structure. This guarantees precise computation of entropy as well as integration with pathfinding algorithms. Two preprocessing techniques are employed: edge weight normalization for normalization of trust values for entropy, graph transformation to transform directed edges to undirected ones with the sum of edge weights. These techniques simplify the complexity of data, get rid of redundancies, and improve the quality of input for entropy-based pruning, and algorithm execution.

2.2.1. Edge weight normalization

Edge weight normalization transforms the original trust values ranging from -10 to +10 into a uniform scale, typically $[0, 1]$, to facilitate entropy calculation. This makes it such that all edge weights contribute to the general information distribution without being skewed towards the entropy results represented as follows.

$$w_{ij}^{norm} = \frac{w_{ij} - w_{min}}{w_{max} - w_{min}}$$

2.2.2. Graph Transformation

The transformation transforms directed edges into undirected ones through averaging scores of mutual trust between node pairs. This process simplifies directional complexity and assists in building a balanced representation of trust to allow for easy analysis and ease of simplifying the graph structure for compatibility with standard algorithms such as Dijkstra's. For each pair of directed edges $(i \rightarrow j)$ and $(j \rightarrow i)$ compute as:

$$w_{ij}^{agg} = \frac{w_{ij} + w_{ji}}{2}$$

3. Proposed Theorem: Graph Edge Entropy Theorem (GEET)

The Graph Edge Entropy Theorem (GEET) proposes a metric to measure the informational importance of each of the edges making up a weighted graph

based on entropy. In this context, entropy measures degree of uncertainty or variability of edge weights, with a higher entropy corresponding to larger information contribution. Let $G = (V, E)$ be a weighted directed graph, where each edge $e_{ij} \in E$ has an associated weight w_{ij} . The probability distribution over all edges weights is defined as:

$$p_{ij} = \frac{w_{ij}}{\sum_{e_{kl}} w_{kl}}$$

The entropy of an edge e_{ij} is defined as:

$$H(e_{ij}) = -p_{ij} \log_2 p_{ij}$$

This helps to capture the information gain from the edges e_{ij} relative to overall structure. A low value of $H(e_{ij})$ indicates minimal contribution to the graph's information complexity. To identify and prune low-entropy edges, a threshold $\theta \in [0, 1]$ is applied:

If $H(e_{ij}) < \theta$, then e_{ij} is pruned

This condition ensured that only edges contributing minimal entropy are removed, preserving key structural and functional properties of the graph.

3.1. GEET-Based Pruning Algorithm

The GEET-based pruning algorithm involves the following steps:

- Normalize all edge weights.
- Compute p_{ij} for each edge using the normalized weight.
- Calculate entropy $H(e_{ij})$ for all edges.
- Remove edges with entropy below the defined threshold θ .

The proposed algorithm is having complexity of $(|E|)$ and thus computationally practical for large graphs.

After pruning, the obtained graph $G' = (V, E')$ is passed to old school algorithms like Dijkstra's. This coupling enables efficient pathfinding with a drastic decrease in computational load without affecting the accuracy of the result.

3.2. Comparative Execution workflow

Both models (original and proposed) are evaluated under controlled conditions. Identical inputs such as source node s , graph structure G and weight configuration are maintained. Key control parameters include:

- Graph size: Total Number of nodes $|V|$
- Energy density Ratio: $D = \frac{2|E|}{|V|(|V|-1)}$
- Pruning threshold θ : Controls the aggressiveness of entropy-based pruning.

Execution time T , memory usage M path accuracy A are recorded for each model:

$$A = \frac{\text{No. of accurate path in } G'}{\text{Total paths in } G} \times 100\%$$

4. Integration with Computational Algorithms

4.1. Algorithm Selection: Dijkstra's Algorithm

Dijkstra's algorithm is a widely used procedure for finding the smallest path between nodes on a weighted graph, having non-negative weights on its edges. Due to the deterministic nature, simplicity and polynomial-time complexity, it is suitable to use it in determining the effects of structural changes such as edge pruning. Dijkstra's algorithm is the baseline for evaluating the effect of the Graph Edge Entropy Theorem (GEET) on the computation time in the course of this study. In traditional model, the algorithm operates on the full graph $G = (V, E)$ where each edge weight is retained as-is. The shortest path from source node s to all other nodes $v \in V$ is computed as:

$$d(v) = \min_{u \in V} \{d(u) + w_{uv}\}$$

In the proposed model a pruned graph $G' = (V, E')$, where $E' \subseteq E$ is used. The reduced edge set results from removing edges with entropy below a defined threshold calculated as:

$$H(e_{ij}) = -p_{ij} \log_2 p_{ij}$$

This modification reduces computational overhead, as fewer edges need to be processed.

4. Result discussion

The results represented in table 1 indicate that the proposed GEET-Dijkstra model significantly improves computational performance compared to traditional and entropy-pruned variants. GEET-Dijkstra achieves the lowest execution time (720 ms) and memory usage (68 MB), demonstrating enhanced efficiency through entropy-based edge pruning. Importantly, it maintains a high path accuracy of 99.2%, only slightly lower than the traditional algorithm, which retains full accuracy at the cost of higher resource consumption. Compared to the standard entropy-pruned model, GEET-Dijkstra offers better accuracy and reduced overhead, validating the effectiveness of the Graph Edge Entropy Theorem in optimizing graph-based computations without sacrificing pathfinding quality.

Table 1
Evaluation parameter of Proposed and traditional approach

Model	Execution Time (ms)	Memory Usage (MB)	Path Accuracy (%)
Traditional Dijkstra's Algorithm	1840	128	100
Entropy-Based Pruned Dijkstra	950	82	98.7
Proposed GEET-Dijkstra	720	68	99.2

5. Conclusion and Future scope

The classical algorithm computational efficiency can be improved greatly by creating new theorems in the graph theory. The present work introduced the Graph Edge Entropy Theorem (GEET) as a new mathematical concept for measuring the informational relevance of edges in weighted graphs. By measuring entropy with normalized edge weights, GEET makes it possible to use principled pruning strategy to simplify the graph while still preserving the crucial structural information. Experimental evaluation over the Bitcoin Alpha trust network showed that usage of GEET with Dijkstra's algorithm brings significant savings on both execution time and memory usage – 61% and 47% respectively – with a high path accuracy of 99.2%. These enhancements validate the balanced approach of optimizing algorithmic performance, preserving accuracy of the output offered by entropy-based graph simplification. GEET's application is scalable, flexible, and compatible with any number of graph-based algorithms besides shortest-path computations. The proposed method serves as an example of how theoretical advances in the graph structures can be implemented in practical computational gains.

References

- [1] S. D. Pasham, Graph-Based Algorithms for Optimizing Data Flow in Distributed Cloud Architectures, pp. 67–95 (2022).
- [2] B. Zhao and G. Guo, "An encryption algorithm for user's sensitive information resources in network database," *Journal of Interdisciplinary Mathematics*, vol. 21, no. 5, pp. 1255–1260 (2018).
- [3] T. Chapuis-Chkaiban, Z. Toffano, and B. Valiron, "On new PageRank computation methods using quantum computing," *Quantum Inf. Process.*, vol. 22, no. 3, pp. 138 (2023).
- [4] C.C. Chou, "A closed-form general solution for the distance of point-to-ellipse in two dimensions," *Journal of Interdisciplinary Mathematics*, vol. 22, no. 3, pp. 337–351 (2019).
- [5] S. Santhanalakshmi, K. Sangeeta, and G. K. Patra, "Design of group key agreement protocol using neural key synchronization," *Journal of Interdisciplinary Mathematics*, vol. 23, no. 2, pp. 435–451 (2020).
- [6] R. Singh, N. Bhardwaj, and K. Alam, "Graph-based recommendation algorithm for personalized suggestions," *J. Discrete Math. Sci. Cryptogr.*, vol. 28, no. 3, pp. 931 (2025).
- [7] T. Zhang, Q. Liao, D. Zhang, C. Zhang, J. Yan, R. Ngetich, J. Zhang, Z. Jin, and L. Li, "Predicting MCI to AD conversion using integrated

- sMRI and rs-fMRI: Machine learning and graph theory approach," *Front. Aging Neurosci.*, vol. 13 (2021).
- [8] F. Shafiei Dizaji and M. Shafiei Dizaji, "Novel computational mathematical algorithms for structural optimization using graph-theoretical methods," *Eng. Comput.*, vol. 39, no. 6, pp. 2391–2423 (Jan. 2022).
- [9] R. Yavari, Z. Smoqi, A. Riensche, B. Bevans, H. Kobir, H. Mendoza, H. Song, K. Cole, and P. Rao, "Part-scale thermal simulation of laser powder bed fusion using graph theory: Effect of thermal history on porosity, microstructure evolution, and recoater crash," *Mater. Des.*, vol. 204, pp. 109685 (2021).

Received April, 2025