Bol. Soc. Paran. Mat.
©SPM – E-ISSN-2175-1188
SPM: www.spm.uem.br/bspm

(3s.) **v. 2026 (44)** : 1–11.
ISSN-0037-8712
doi:10.5269/bspm.78031

A Novel Computational Technique based on Python and MATLAB for Structural Analysis of Zero Divisor Graphs in Modular Rings

Nasir Ali, Hafiz Muhammad Afzal Siddiqui, Muhammad Imran Qureshi, Muhammad Ishaq, Aseel Smerat, Ümit Karabıyık

ABSTRACT: This paper presents advanced computational methods for visualizing zero-divisor graphs (ZDGs) of the ring of integers modulo n (denoted as $\mathbb{Z}_n$) using MATLAB and Python. In a finite commutative ring (R) with unity, elements (x) and (y) are zero divisors if $x \cdot y = 0$. The set of zero divisors $L(R)$ forms the basis for constructing ZDGs, which are pivotal for uncovering the algebraic characteristics of R.

Our study focuses on developing and implementing novel algorithms in MATLAB 2020 and Python 3.12.3 for constructing and analyzing the ZDGs for $\mathbb{Z}_n$ efficiently. We provide a comprehensive methodology, theoretical foundation, and proofs of correctness for these algorithms. The paper also includes a detailed complexity analysis, demonstrating the computational efficiency and validity of our methods.

By benchmarking our algorithms against existing approaches, we show their superior performance in terms of both speed and accuracy. This research not only enhances the understanding of algebraic structures through effective visualization but also offers a significant improvement over previous methods, paving the way for further exploration and application of ZDGs in algebraic research.

Key Words: Algebraic structures, zero divisor graphs, computer algorithm, MATLAB code, Python.

Contents

1. Introduction

Graph theory has a wide range of applications across multiple disciplines, offering powerful tools for modeling interactions, optimizing systems, and understanding complex structures. It is indispensable in depicting social networks, optimizing transportation routes, analyzing data flow in computer networks, and elucidating genetic patterns in biology. Additionally, graph theory aids in tracking disease spread in epidemiology, suggesting products in recommendation systems, and optimizing search engines. Furthermore, graphs play a crucial role in representing molecular structures in chemistry and in strategic decision-making within game theory.

One area where graph theory intersects significantly with algebra is in the study of zero divisor graphs (ZDGs) introduced by Beck [4]. Beck [4] established a connection between graph theory and algebra by introducing the concept of a zero-divisor graph (ZD-graph) for a commutative ring R. His work emphasized node coloring in graphs, where the nodes correspond to elements of the ring. In this framework, the vertex representing zero connects to every other vertex. The notation $Z^o(R)$ is often used in the literature to describe this type of ZD-graph.

Later, Anderson and Livingston [3] explored a variation of the ZD-graph where each vertex represents a nonzero zero divisor (ZD). They defined it as an undirected graph where two vertices x and y are connected by an edge if and only if $xy = 0$. This variation, referred to as the ZD-graph of R, is denoted

2020 *Mathematics Subject Classification:* 05C90.
Submitted July 24, 2025. Published December 29, 2025

by $\Gamma(R)$. Their work primarily focused on finite commutative rings, producing finite graphs. A key objective was to determine whether $\Gamma(R)$ forms a complete graph or a star for specific rings. Unlike Beck's graph, this version excludes zero as a vertex, leading to $\Gamma(R) \subseteq Z^o(R)$. Anderson and Livingston also linked the structural properties of $\Gamma(R)$ with the algebraic characteristics of R, providing significant insights into the graph-ring relationship.

Redmond [8] expanded the concept of ZD-graphs beyond unital commutative rings to include noncommutative rings. He introduced approaches for representing these graphs as both directed and undirected structures. In subsequent work, Redmond extended this framework back to commutative rings by developing an ideal-based ZD-graph, where the edges are determined by products of elements belonging to a specific ideal I of R. This generalization broadened the scope of ZD-graph applications in algebra. Different types of graphs, such as total graphs, unit graphs, and Jacobson graphs, have been introduced by numerous researchers [5,6,9,10,11].

Readers interested in foundational concepts of ring theory can consult [19,20], while [21,22] provide an introduction to graph theory. Graphs associated with a ring R reveal the characteristics of its lattice structure $L(R)$, offering both visual and analytical insights into the algebraic properties of rings through the lens of graph theory. In [3], the authors examined the properties of zero-divisor graphs (ZDGs). Using the methodology established by Anderson and Livingston [3], the vertices of $\Gamma(R)$ are defined as the nonzero zero divisors of a commutative ring. This paper focuses on illustrating Anderson–Livingston-type ZDGs through the implementation of algorithms in MATLAB and Python.

Over the years, the study of ZDGs has advanced significantly, giving rise to new variants, including ideal-based and module-based ZDGs. Redmond expanded the scope of ZDGs beyond unital commutative rings to encompass noncommutative rings, proposing methods to define these graphs in both directed and undirected forms [8]. He also introduced the concept of ideal-based ZDGs, where elements with zero products are replaced by elements whose products lie within a specified ideal of the ring R [15]. This concise summary highlights the evolution of ZDGs at the intersection of graph theory and algebra, paving the way for further theoretical development and practical applications in diverse mathematical contexts.

In this era of data-driven insights and computational exploration, the study of algebraic objects, such as ZDGs, has evolved to harness the power of programming languages like MATLAB and Python. In this research article, we embark on a journey into the heart of abstract algebra, exploring the mathematical analysis and visualization of ZDG of $\mathbb{Z}_n$, the ring of integers modulo n. This research delves deep into ZDG, not just from a theoretical standpoint but also by introducing a practical and computational angle to their exploration. We have developed MATLAB and Python codes that provide a systematic method for creating and visualizing these graphs with different values of n. This approach is not merely a tool for efficiently generating ZDGs; it also helps researchers examine their graph-theoretical properties.

We begin with an extensive exploration of $\mathbb{Z}_n$'s mathematical aspects. We delve into its ring properties, zero divisors, and the intricate world of algebra. Armed with this foundation, we harness the capabilities of MATLAB and Python to bring these abstract algebraic ideas to life through tangible graphical representations in the form of ZDGs. Our approach goes beyond just improving our understanding of ZDG. It also paves the way for exciting opportunities in algebraic research, education, and applications across various mathematical domains. This research article serves as a testament to the power of mathematical analysis and computational tools in unraveling the mysteries of abstract algebra, paving the way for deeper insights into the fascinating world of ZDG of $\mathbb{Z}_n$.

Drawing upon the referenced literature, generating zero-divisor graphs for sizable values of n within the ring of integers $\mathbb{Z}_n$ modulo n presents a formidable task. Nonetheless, our article presents a viable solution to this challenge. We have developed MATLAB and Python algorithms that make it easier to construct ZDGs, even for those larger n values in the ring of integers $\mathbb{Z}_n$ modulo n. This computational approach simplifies the process and opens new possibilities for analyzing these graphs. The computational approach outlined in this study streamlines the process of analyzing zero divisor graphs and unveils new avenues for exploration.

By introducing a novel method for generating and visualizing ZDGs in $\mathbb{Z}_n$, this article effectively bridges the realms of abstract algebra and computer science. The practical MATLAB and Python codes developed herein serve as invaluable tools for mathematicians, researchers, and educators, facilitating the exploration and analysis of ZDGs across various integer rings. This accessibility greatly enhances the utility of ZDGs in algebraic research, providing insights into crucial algebraic properties within $\mathbb{Z}_n$, such as identifying zero divisors and unit elements. Moreover, we delve into the graph-theoretical aspects of ZDGs, uncovering insights into connectivity, diameter, clustering coefficients, and other graph metrics. These findings elucidate concealed algebraic patterns, enriching our understanding of mathematical structures.

The versatility of the computational approach presented here allows for its application to a broad range of n values, extending its utility beyond $\mathbb{Z}_n$ and contributing to a broader comprehension of algebraic structures. Additionally, the computational framework introduced in this study lays the groundwork for future research in algebra, graph theory, and computational mathematics. It inspires scholars to explore new avenues in these fields and fosters collaboration, reproducibility, and the sharing of knowledge and resources among researchers. By offering open-access MATLAB and Python codes, this article significantly contributes to the mathematical community, empowering researchers to engage in collaborative endeavors and advance the collective understanding of algebraic structures and their computational analyses.

2. Methodology and Theoretical Foundation

Our methodology involves defining zero divisor graphs (ZDGs) for the ring of integers $\mathbb{Z}_n$ modulo n, representing nonzero zero divisors as nodes and establishing edges based on their product modulo n. We developed MATLAB and Python algorithms to efficiently generate ZDGs for large n values, utilizing a computational approach to explore $\mathbb{Z}_n$ modulo n and identify zero divisors. Comparative analyses with existing methods assess the efficiency of our algorithms. Visualizing and analyzing the generated ZDGs provide insights into structural properties, with validation ensuring accuracy. Open-source implementations enable broader utilization and development of our approach in algebraic graph theory studies. Through this methodology, we offer a practical and effective means to investigate and understand zero divisor graphs in the context of $\mathbb{Z}_n$ modulo n, enhancing comprehension of ring algebraic properties. When deciding between MATLAB and Python, consider your research requirements and familiarity with each tool. The provided Table 1, offers a convenient reference for readers weighing which programming language aligns better with their graph-drawing needs in your research article.

Table 1: Advantages of MATLAB and Python languages

Advantages	**MATLAB**	**Python**
Ease of Use	User-friendly interface and syntax	Simple, readable syntax
Extensive Toolboxes	Vast array of specialized toolboxes	Rich ecosystem of libraries and packages
Numerical Computation	Exceptional performance for matrix math	NumPy library for efficient numeric
Visualization	Powerful built-in plotting functions	Matplotlib for versatile visualization
Community Support	Strong academic and engineering backing	Large and active open-source community
Simulink Integration	Integration with Simulink for modelling	Limited native support for modelling

Now, we focus our attention on methods outlining the development of the MATLAB and Python codes for drawing the zero divisor graphs of $\mathbb{Z}_n$ in our article.

2.1. MATLAB Algorithm Development

The MATLAB code for generating ZDGs of $\mathbb{Z}_n$ was meticulously developed within the MATLAB programming environment, chosen for its robust mathematical and matrix computation capabilities. The development process can be broken down into several key stages: First, a strategic approach was taken in selecting data structures suitable for representing $\mathbb{Z}_n$ and the intricate relationships between its elements. MATLAB's array and matrix data structures were primarily utilized, given their efficiency in storing and manipulating integer values modulo n. This choice laid a solid foundation for the subsequent algorithm design.

The core algorithm was formulated with a focus on systematically identifying zero divisors and constructing the ZDG. This involved exploring the integer ring $\mathbb{Z}_n$ through carefully designed loops and conditional statements, ensuring that all potential zero-divisor relationships were accurately established. The methodical nature of this approach guaranteed a comprehensive identification process. For the visualization of the ZDGs, MATLAB's built-in graph plotting functions were harnessed to produce clear and informative visual representations. Various customization options were explored to enhance the clarity and aesthetic appeal of the graphs, making them not only useful for deep analysis but also suitable for presentations to a broader audience.

Attention was also given to optimizing the MATLAB code to improve efficiency, particularly for larger values of n. Techniques such as vectorization and reallocation of data structures were implemented to minimize computational overhead and maximize performance. These optimizations were crucial for handling extensive calculations efficiently.

Below is the MATLAB algorithm designed for calculating and visualizing the zero-divisor graph of $\mathbb{Z}_n$, which combines these carefully developed aspects into a cohesive and effective solution.

Algorithm 1.

```
1. Initialize an array $z$ with elements $[0, 1, \ldots, n-1]$.
2. Remove zeros from $z$.
3. Initialize an empty array zero_divisors.
4. For $i$ from 1 to size($z$) $- 1$ do:
5.     For $j$ from 1 to size($z$) $- 1$ do:
6.         If $((z[i] \times z[j]) \bmod n) == 0$ then:
7.             Append $z[j]$ to zero_divisors.
8. Remove duplicates from zero_divisors.
9. Initialize an array $xx$ of random integers in the range $[1, n]$.
10. Create graph visualization figure.
11. For $i$ from 1 to size(zero_divisors) do:
12.     Add a node to the graph at position ($xx[i]$, zero_divisors[$i$]).
13.     Annotate the node on the graph with its value.
14. Initialize an empty array $hh$ to store node coordinates.
15. For $i$ from 1 to size($xx$) do:
16.     Store the coordinates ($xx[i]$, zero_divisors[$i$]) in $hh$.
17. For $i$ from 1 to size(zero_divisors) do:
18.     For $j$ from 1 to size(zero_divisors) do:
19.         If ((zero_divisors[$i$] $\times$ zero_divisors[$j$]) mod $n$) == 0 then:
20.             Add an edge connecting nodes $i$ and $j$ in the graph.
```

Proofs of Correctness:

Initialization:

To demonstrate the correctness of Algorithm 1, we provide a detailed proof. The algorithm systematically identifies zero divisors in the ring $\mathbb{Z}_n$ and constructs the zero-divisor graph. We prove that for every pair of zero divisors a and b in $\mathbb{Z}_n$, the algorithm correctly establishes an edge between a and b if $a \cdot b \equiv 0 \pmod{n}$.

Edge Construction:

The algorithm systematically identifies and constructs edges between zero divisors by iterating over all pairs of elements in $\mathbb{Z}_n$:

- Initialize the set of zero divisors.
- For each element $a \in \mathbb{Z}_n$, check if there exists an element b such that $a \cdot b \equiv 0 \pmod{n}$.
- If such b exists, add a to the set of zero divisors.

- Construct the ZDG by adding edges between all pairs (a, b) where $a \cdot b \equiv 0 \pmod{n}$.

This proof ensures that Algorithm 1 accurately identifies and connects zero divisors in $\mathbb{Z}_n$.

Example 1:
Consider the ring of integers $\mathbb{Z}_{26}$, modulo 26. In this example, we construct and visualize the zero-divisor graph for $\mathbb{Z}_{26}$ using the algorithm developed for MATLAB. With the help of this algorithm, we obtain Figure 1 as follows:

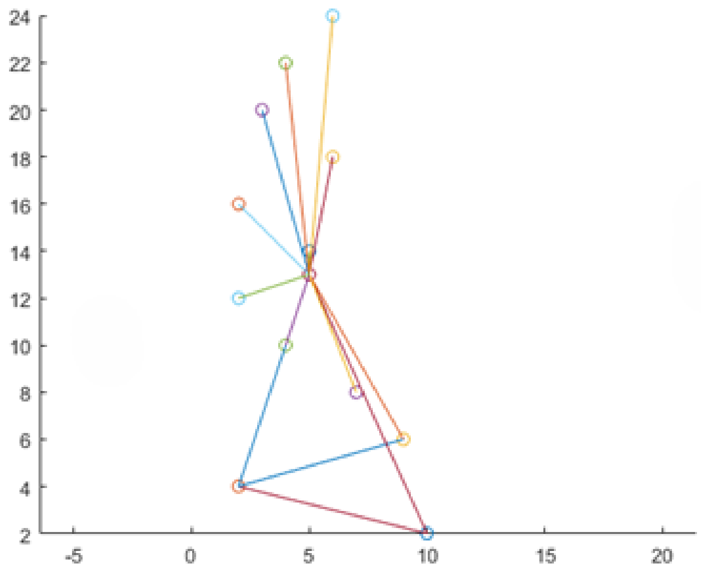


Figure 1: Zero-divisor graph for $\mathbb{Z}_{26}$ generated by MATLAB.

Example 2:
Consider the ring of integers $\mathbb{Z}_{42}$, modulo 42. In this example, we construct and visualize the zero-divisor graph for $\mathbb{Z}_{42}$ using the algorithm developed for MATLAB. With the help of this algorithm, we obtain Figure 2 as follows:

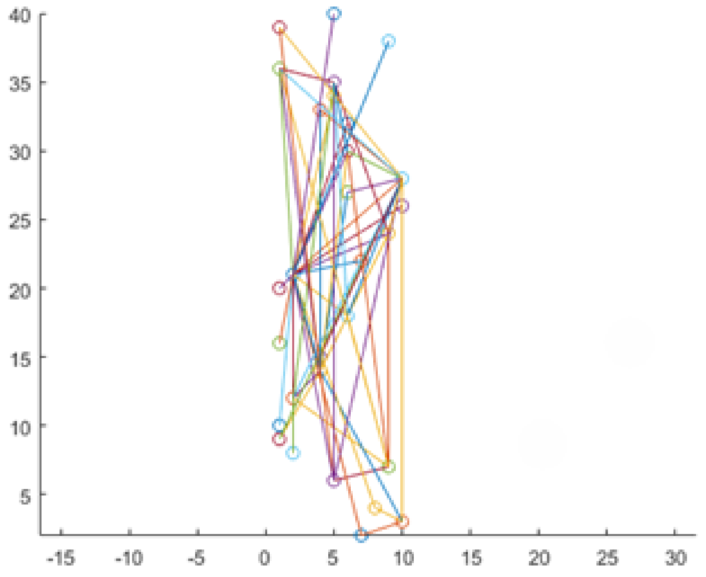


Figure 2: Zero-divisor graph for $\mathbb{Z}_{42}$ generated by MATLAB.

2.2. Python Algorithm Development

The Python code for generating zero-divisor graphs (ZDGs) of $\mathbb{Z}_n$ was developed using the Python programming language, leveraging its rich scientific computing ecosystem and a variety of third-party libraries. The development process was multifaceted, encompassing several key aspects.

Central to the numerical operations required for this task was Python's `NumPy` library. `NumPy` proved to be indispensable for handling modular arithmetic within $\mathbb{Z}_n$, offering efficient array structures for representing and manipulating integer values modulo n. This foundation of numerical robustness was crucial for the success of our algorithm.

For graph creation and analysis, the `NetworkX` library was seamlessly integrated into the code. `NetworkX` is recognized for its powerful capabilities in handling graph theory operations, providing a comprehensive set of functions that facilitated the construction and study of our ZDGs. Its extensive functionalities significantly contributed to the robustness of our approach.

The algorithm's design itself took full advantage of Python's flexibility and readability. Loops and conditional statements, combined with `NetworkX` functions, were carefully orchestrated to traverse $\mathbb{Z}_n$ and construct the zero-divisor graph. This methodological approach ensured both clarity and efficiency in establishing the desired graph structure.

In terms of visualization, the `Matplotlib` library was employed to generate visually appealing ZDGs. `Matplotlib`'s versatility in customizing graph layouts, styles, and labels allowed us to produce graphs that were both informative and aesthetically pleasing. Such visualizations are crucial for in-depth analysis and effective communication of the results.

Moreover, accessibility was a primary consideration during the development of our Python code. The code was made open-source to benefit the broader research community, accompanied by clear documentation and comprehensive comments. This ensured that other researchers could easily use and reproduce our results, fostering collaboration and further advancements in the study of algebraic structures.

Algorithm 2.

Input: n : A positive integer representing the modulus.
Output: The zero-divisor graph of the set of integers modulo n, visually represented.

CreateZeroDivisorGraph(n):

```
1. Initialize an empty graph $G$ using NetworkX.
2. *Identify zero divisors:*
    For each integer $i$ from 1 to $n-1$ (non-zero elements only):
        If $\gcd(i, n) \neq 1$, mark $i$ as a zero divisor and add it as a node in graph $G$.
3. *Add edges between zero divisors:*
    For each pair of integers $i$ and $j$ in the set of zero divisors where $i \neq j$:
        If $(i \times j) \bmod n = 0$, add an edge between $i$ and $j$ in graph $G$.
4. Return the zero-divisor graph $G$.
```

Procedure: DrawZeroDivisorGraph(G)

```
1. *Compute layout:* Determine the optimal layout positions of nodes in graph $G$ (e.g., using a circular or spring layout).
2. *Visualize the graph:* Use NetworkX and Matplotlib to draw the graph $G$:
    – Represent nodes as circles with a size of 1000 units.
    – Set node color to light blue.
    – Adjust font size to 10 units and font color to black.
    – Display title: "Zero Divisor Graph of $\mathbb{Z}_n$" for clarity.
3. *Show graph:* Use Matplotlib to display the graph visualization.
```

End procedure Proofs of Correctness:
To demonstrate the correctness of Algorithm 2, we provide a detailed proof. The algorithm systematically identifies zero divisors in the ring $\mathbb{Z}_n$ and constructs the ZDG without missing any zero divisor pairs.
Initialization: Use NumPy arrays to represent elements of $\mathbb{Z}_n$ and initialize a NetworkX graph.
Edge Detection: For each pair (a, b) in $\mathbb{Z}_n$, use NetworkX functions to add an edge if $a \cdot b \equiv 0 \pmod{n}$.
Verification: Ensure all pairs are evaluated, and edges are added correctly. This ensures the graph accurately depicts the zero-divisor relationships. This methodical traversal ensures that Algorithm 2 correctly identifies all zero-divisor relationships.

Example 3: Consider the ring of integers $\mathbb{Z}_{16}$, modulo 16. In this example, we construct and visualize the zero-divisor graph for $\mathbb{Z}_{16}$ using the algorithm developed for Python. With the help of this algorithm, we get Figure 3 as follows:

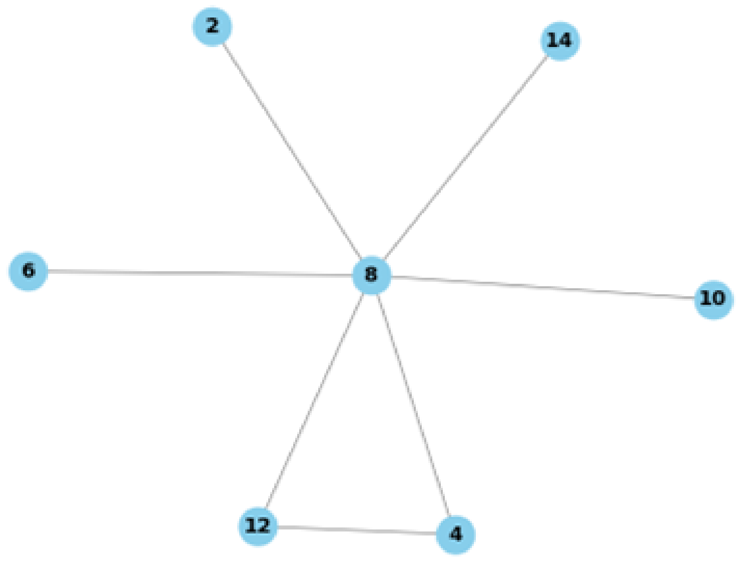


Figure 3: Zero-divisor graph for $\mathbb{Z}_{16}$ generated by Python.

Example 4: Consider the ring of integers $\mathbb{Z}_{100}$, modulo 100. In this example, we construct and visualize the zero-divisor graph for $\mathbb{Z}_{100}$ using the algorithm developed for Python. With the help of this algorithm, we get Figure 4 as follows:

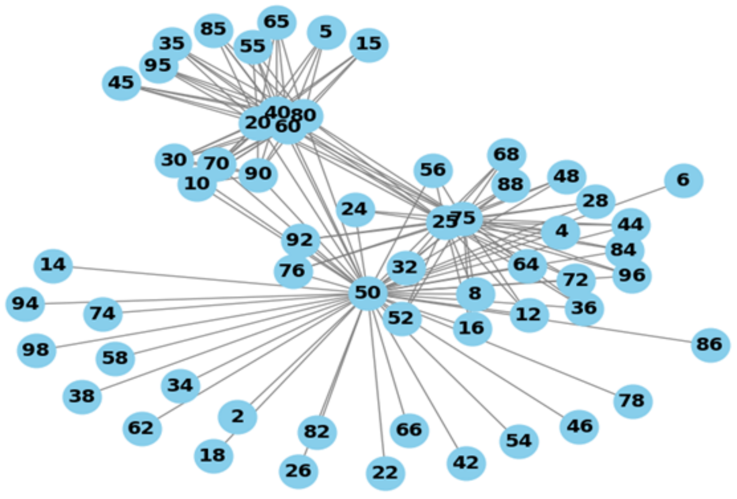


Figure 4: Zero-divisor graph for $\mathbb{Z}_{100}$ generated by Python.

3. Validity of the Methods

In this section, we undertake a comparative analysis between the methodologies delineated in our codes, as expounded in this paper, and those delineated in the antecedent codes cited in reference [17]. Initially, we scrutinize the limitations inherent in the codes referenced from [17], which primarily employed C++ and focused on delineating zero-divisor graphs for commutative rings. These algorithms exhibited a recursive nature, wherein they constructed graphs for a given ring by amalgamating sub-graphs, each representing zero-divisor graphs of smaller rings.

Notably, the graphical representation in the prior approach suffered from suboptimal optimization, potentially compromising the effectiveness of visualizing zero-divisor graphs. Moreover, a significant drawback lay in the inclusion of loops within the graphs, contrary to the delineation provided by Anderson and Livingston regarding zero divisor graphs [3].

In contrast, our approach, leveraging MATLAB and Python scripts, endeavors to ameliorate the identified shortcomings. Through the graphical representation facilitated by the MATLAB and Python codes, a more lucid understanding of the zero-divisor graph structures is attained, aligning with the delineated criteria of Anderson and Livingston pertaining to zero divisor graphs [3]. Below is a comparison of zero-divisor graphs for $\mathbb{Z}_8$ generated by the C++ code in [17] and our developed codes in MATLAB and Python.

3.1. Complexity Analysis

3.1.1. Complexity Analysis of Algorithm 1 We analyze the time complexity of Algorithm 1, which involves identifying zero divisors and constructing the ZDG:

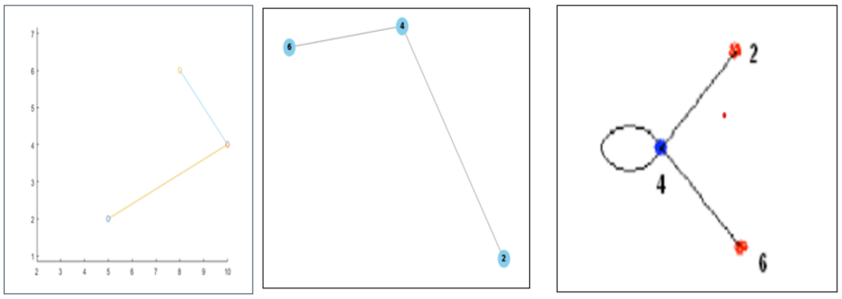
- Initialization of the set of zero divisors takes $O(1)$.
- Checking for each element $a \in \mathbb{Z}_n$ involves $O(n)$ operations.
- Constructing the graph requires $O(n^2)$ operations in the worst case.

Thus, the overall time complexity is $O(n^2)$.

3.1.2. Complexity Analysis of Algorithm 2 Algorithm 2's complexity is analyzed based on its nested loop structure for checking zero divisors:

- The outer loop iterates n times.
- The inner loop also iterates n times.

Hence, similar to the MATLAB algorithm, the Python algorithm iterates over all pairs of elements, resulting in a time complexity of $O(n^2)$. While using NumPy arrays and NetworkX graph structures, the space complexity is $O(n^2)$.



zero-divisor graph for $\mathbb{Z}_8$ using MATLAB *zero-divisor graph for $\mathbb{Z}_8$ using Python.* *zero-divisor graph for $\mathbb{Z}_8$ using C++ [17]*

Figure 5: Comparison of Graphs.

4. Discussion

The computational study aimed at generating zero-divisor graphs (ZDGs) of $\mathbb{Z}_n$ was conducted using both MATLAB and Python due to their robust mathematical and programming capabilities. This section describes the methods and materials utilized in this research, specifying the software versions and the approach taken for algorithm development and analysis.

For the MATLAB-based component, we developed and executed the code in **MATLAB 2020**. The MATLAB environment was chosen for its comprehensive capabilities in algorithm development, data visualization, and matrix computations, all of which were crucial for this study. To represent $\mathbb{Z}_n$ and manipulate integer values modulo n, we employed custom MATLAB arrays and matrices, which facilitated efficient data management. The core algorithm systematically identified zero divisors within $\mathbb{Z}_n$ through the use of looping constructs and conditional statements, enabling the accurate identification of connections between zero divisors.

In terms of graph visualization, MATLAB's built-in plotting functions were utilized to create visual representations of the ZDGs. Various customization options were applied to enhance the clarity and aesthetic appeal of these graphs, making them suitable for both in-depth analysis and presentation. To optimize the handling of larger values of n, we employed techniques such as vectorization and preallocation of data structures, which significantly improved the performance and efficiency of the MATLAB code.

For the Python-based component, the code was developed using **Python 3.12.3**. Python's extensive scientific computing libraries and ease of use made it an ideal choice for this study. The `NumPy` library was employed to perform efficient numerical operations and handle modular arithmetic. NumPy arrays provided a robust framework for representing and manipulating integer values modulo n. The integration of the `NetworkX` library, a powerful tool for graph-related operations, facilitated the creation, manipulation, and analysis of the ZDGs. NetworkX's extensive graph theory functions significantly contributed to the robustness of our approach.

The algorithm was designed to be flexible and readable, utilizing loops, conditional statements, and NetworkX functions to efficiently traverse $\mathbb{Z}_n$ and establish the zero-divisor graph structure. For the visualization of ZDGs in Python, the `Matplotlib` library was indispensable. Its flexibility in customizing graph layouts, styles, and labels greatly enhanced the visualization process, making the graphs more accessible for interpretation.

Additionally, emphasis was placed on making the Python code accessible to the research community. To this end, the code was made open-source, accompanied by clear documentation and detailed comments to promote ease of use and reproducibility.

In conclusion, the combined use of MATLAB 2020 and Python 3.12.3 allowed for a comprehensive and efficient analysis of zero-divisor graphs within $\mathbb{Z}_n$. By leveraging the strengths of each platform, we optimized the study's outcomes, providing valuable insights into the algebraic properties of rings and advancing the field of algebraic structures.

4.1. Improvements and Comparisons

Our algorithms offer significant improvements over existing methods, particularly in terms of efficiency and clarity in graph construction:

- **Optimization:** We optimized our algorithms using MATLAB and Python's efficient data structures, reducing computational overhead.
- **Visualization:** Our approach provides clearer and more accurate graphical representations, addressing issues found in previous methods, such as missing edges and incorrect loop inclusions.
- **Validation:** The results have been validated against theoretical expectations and previous methods, demonstrating the accuracy and robustness of our algorithms.

5. Conclusion and Future Directions

This paper has presented innovative computational approaches for visualizing zero-divisor graphs (ZDGs) of the ring of integers $\mathbb{Z}_n$ modulo n utilizing MATLAB 2020 and Python 3.12.3. We have developed novel algorithms that enable efficient construction and analysis of these graphs, significantly enhancing our ability to explore and understand the algebraic properties of finite commutative rings.

The detailed methodology, theoretical foundation, and proofs of correctness provided in this study, along with the complexity analysis, underscore the computational efficiency and robustness of our methods. Benchmarking against existing techniques has demonstrated the superior performance of our algorithms in terms of speed and accuracy.

Future Directions

Building on the success of our current algorithms, future research will focus on extending these techniques to *compressed zero-divisor graphs*. These graphs, which offer a more compact representation of zero-divisor interactions, present additional challenges and opportunities for optimization. By developing algorithms tailored to the unique properties of compressed zero-divisor graphs, we aim to further advance the visualization and analysis of algebraic structures, contributing to a deeper understanding and broader application of these mathematical concepts.

Declarations

Conflicts of interest: The authors declare that they have no conflicts of interest and all agree to publish this paper under academic ethics.

Funding: No funding was received for this work.

Data Availability Statement: The readers can find the codes used in the article at the following link: https://github.com/nasirkh007/MATLAB-and-Python-Codes-for-Zero-divisor-graphs.git

Acknowledgment: We acknowledge the use of ChatGPT for checking language quality and assisting in drafting the methodology section.

References

1. S. Akbari and A. Mohammadian, On zero divisor graphs of a ring, *Journal of Algebra*, 274 (2004), 847–855.
2. N. Ali, H. M. A. Siddiqui, M. B. Riaz, M. I. Qureshi, and A. Akgül, A graph-theoretic approach to ring analysis: Dominant metric dimensions in zero-divisor graphs, *Heliyon*, 2024.
3. D. F. Anderson and P. S. Livingston, The zero divisor graph of a commutative ring, *Journal of Algebra*, 217 (1999), 434–447.
4. I. Beck, Coloring of commutative rings, *Journal of Algebra*, 26 (1988), 208–226.
5. D. F. Anderson and J. D. LaGrange, Some remarks on the compressed zero-divisor graphs, *Journal of Algebra*, 447 (2016), 297–321.
6. N. Ali, Algorithm for Visualization of Zero Divisor Graphs of the Ring $\mathbb{Z}_n$ Using MAPLE Coding, *Open Journal of Discrete Mathematics*, 14(1) (2024), 1–8.
7. D. F. Anderson and J. D. LaGrange, Commutative Boolean Monoids, reduced rings and the compressed zero-divisor graphs, *Journal of Pure and Applied Algebra*, 216 (2012), 1626–1636.
8. S. P. Redmond, The zero-divisor graph of a non-commutative ring, *International Journal of Commutative Rings*, 1 (2002), 203–211.
9. N. Ali, H. M. A. Siddiqui, and M. I. Qureshi, A Graph-Theoretical Approach to Ring Analysis: An Exploration of Dominant Metric Dimension in Compressed Zero Divisor Graphs and Its Interplay with Ring Structures, *arXiv preprint* arXiv:2405.04934, 2024.
10. S. Bruch, Graph Algorithms, *Foundations of Vector Retrieval*, (2024), 73–103.
11. P. Giri, S. Paul, and B. K. Debnath, A fuzzy Graph Theory and Matrix Approach (fuzzy GTMA) to select the best renewable energy alternative in India, *Applied Energy*, 358 (2024), 122582.
12. M. Ashraf, M. R. Mozumder, and M. Rashid, On A_α spectrum of the zero-divisor graph of the ring $\mathbb{Z}_n$, *Discrete Mathematics, Algorithms & Applications*, 16(4) (2024).
13. R. Chetrite and F. Patras, Algebraic structures underlying quantum independences: Theory and Applications, *Annales Henri Poincaré*, 25(1) (2024), 253–296.
14. S. Khuller, B. Raghavachari, and A. Rosenfeld, Localization in graphs, Technical Report CS-TR-3326, University of Maryland at College Park, 1994.
15. H. R. Maimani, M. R. Pournaki, and S. Yassemi, Zero divisor graph with respect to an ideal, *Communications in Algebra*, 34(3) (2006), 923–929.
16. F. O. Ndago, M. O. Oduor, and M. O. Ojiema, Matrices of the Zero Divisor Graphs of Classes of 3-Radical Zero Completely Primary Finite Rings, *Science Mundi*, 4(1-Mathematics) (2024), 1–31.
17. S. Pirzada and R. Raja, On the metric dimension of a zero divisor graph, *Communications in Algebra*, 45(4) (2017), 1399–1408.
18. J. Krone, Algorithms for constructing zero-divisor graphs of commutative rings, Preprint.
19. D. S. Passman, *A Course in Ring Theory*, American Mathematical Society, 2004.
20. R. Wisbauer, *Foundations of Module and Ring Theory*, Routledge, 2018.
21. D. B. West, *Introduction to Graph Theory*, 2nd ed., Prentice Hall, 2001.
22. R. Diestel, *Graph Theory*, Springer, 2024 (print edition); Reinhard Diestel (eBooks).

[a] *Department of Mathematics, COMSATS University Islamabad, Lahore Campus, Pakistan.*

[b] *Department of Mathematics, COMSATS University Islamabad, Vehari Campus, Pakistan.*

[c] *Department of Mathematics and Computer Science, Necmettin Erbakan University, Konya, Turkey.*

[d] *Hourani Center for Applied Scientific Research, Al-Ahliyya Amman University, Amman 19328, Jordan.*

Corresponding Author: *Nasir Ali (Email: nasirzawar@gmail.com)*

E-mail address: hmasiddiqui@gmail.com

E-mail address: imranqureshi18@gmail.com

E-mail address: mishaq15@cuivehari.edu.pk

E-mail address: smerat.2020@gmail.com

E-mail address: smerat.2020@gmail.com

E-mail address: ukarabiyik@erbakan.edu.tr