Int. J. Anal. Appl. (2025), 23:26

International Journal of Analysis and Applications

A Computational Approach towards Fragment Based Drug Design and Analysis Using G = (V, E) Decomposition

M. Sivasankari, M. Yamuna[*]

Department of Mathematics, SAS, Vellore Institute of Technology, Vellore, TamilNadu, India

[*]*Corresponding author:* myamuna@vit.ac.in

Abstract. Objectives: To develop an efficient algorithm for decomposing arbitrary graphs into circuits and paths, thereby enabling a more comprehensive analysis of molecules for fragment-based drug design. Methods: We have devised Algorithm GD for decomposing any graph into its constituent circuits and paths. A MATLAB implementation of this algorithm was developed to generate the necessary outputs. Algorithm GD was applied to identify non-overlapping fragments within drug molecules. Results: The MATLAB code's performance was evaluated in terms of sample outputs and runtime calculations. Algorithm GD was successfully employed to determine the non-overlapping fragments of fungicides. Subsequently, the Wiener Index of these fragments was calculated. Conclusion: A regression equation was established between the graph Wiener Index estimated from non-overlapping fragments and log KOC values. This model can be utilized to predict the log KOC values of fungicides without the need for experimental setups, thereby streamlining the drug discovery process.

1. Introduction

Decomposing graphs involves breaking down complex structures into simpler building blocks. Research in graph decomposition explores various approaches, including decomposing regular graphs, directed graphs, and complex multipartite graphs with ascending subgraph structures ([1], [2]). Decomposition helps in crystal structure prediction, defining new class of graph through vertex neighbourhoods ([3], [4]). New approaches for breaking down G into paths, trees and cycles were developed for various type of graphs ([5], [6], [7], [12]). The relationship between decomposing graphs by preserving specific connectivity properties and inherent non-adjacency properties within a graph, decomposing complete graphs into fork graphs, tensor graphs into cycles and star graphs etc are also discussed by authors ([8], [9], [10], [11]).

Received: Sep. 9, 2024.

2020 *Mathematics Subject Classification.* 05C90, 05C92.

Key words and phrases. circuits; decomposition; FBDD; fungicides; paths.

https://doi.org/10.28924/2291-8639-23-2025-26

ISSN: 2291-8639

Some studies focus on constructing larger graphs entirely from smaller ones, like paths or stars, with a specific number of edges. For instance, one approach investigates the number of edges needed to build a graph only from paths, stars and crown graphs, and other research tackles the efficient decomposition of planar graphs ([13], [14], [15]). Generalizing the concept further, some research tackles decomposing a special class of graphs called NC-graphs into two subgraphs [16]. After reviewing the literature on graph decomposition, we observed that classical methods often rely on the symmetry of the underlying graph structures. This raises the following questions

(1) Can we decompose only graphs with specific symmetries, or can we decompose any given graph G with n vertices into known structures?
(2) Can we develop algorithms to generate outputs for the decomposition?
(3) Can we apply these algorithms to various domains?

This article focuses on answering these questions. The rest of the paper is organized as followed in Graphical Representation.

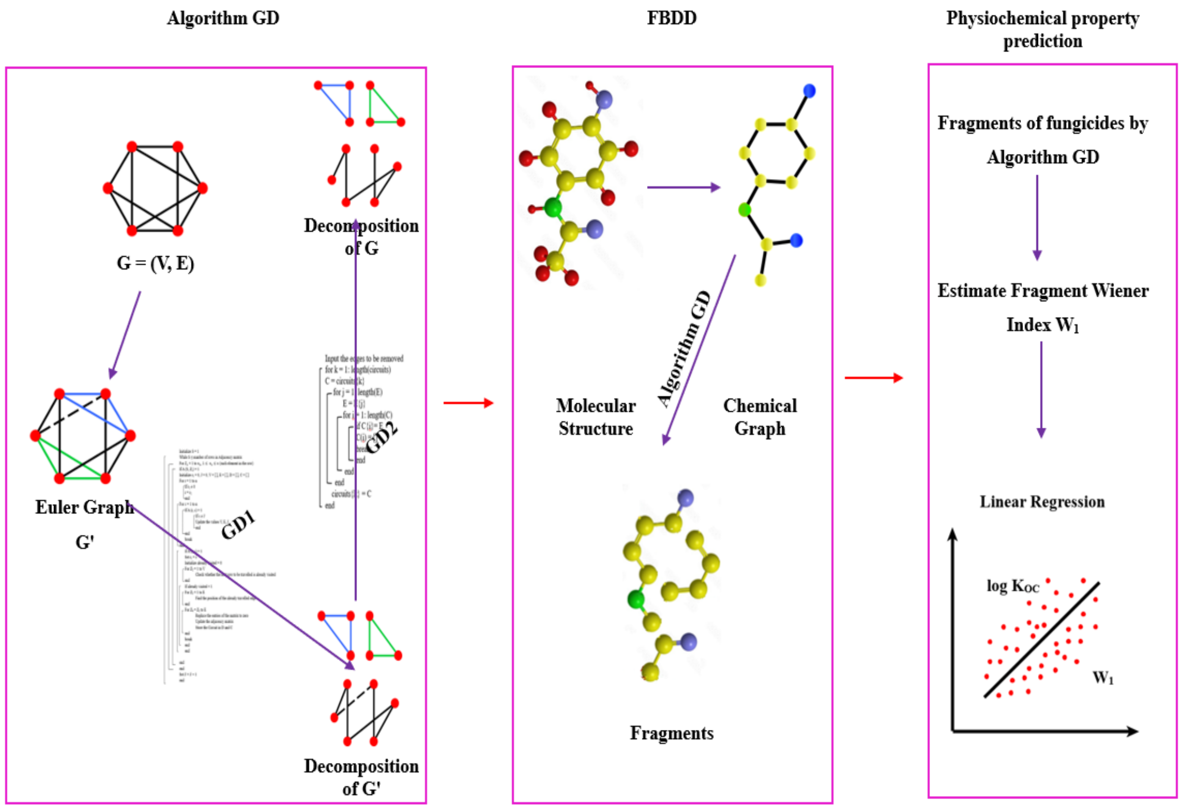


FIGURE 1.

2. G = (V, E) Decomposition

2.1. **Decomposition into Circuits and Paths.** In this section we provide a solution for Question 1. To achieve this, we focused on existing results in graph theory that are applicable to our graph with n vertices. We recalled the following famous theorems related to Euler graphs:
R$_1$: A connected graph G is an Euler graph if and only if all vertices of G are of even degree.
R$_2$: A connected graph G is an Euler graph if and only if it can be decomposed into circuits.
The next question that can be posed is what if G is non-Euler? In this angle we picked the following result.
R$_3$: In a connected graph G with exactly $2k$ odd vertices, there exist k edge-disjoint subgraphs such that they together contain all edges of G and that each is a unicursal graph.
The proof technique of R_3 results in a characterization of a non-Euler graph into unicursal graphs using an Euler line of a constructed Euler graph. This gives a clue that a non-Euler graph can be constructed into an Euler graph by adding edges. In this article we decided to use R_1, R_2, and R_3. To make this possible we shall use adjacency matrix, since a graph is best represented as a matrix using adjacency matrix. The matrix representation is efficient for computational purpose. The representation of the adjacency between vertices and the symmetry of the adjacency matrix can be used for developing an algorithm to decompose any Euler graph G into circuits as R_2 states. In case of non-Euler graphs R_3 can be used for the same. With all these observations, in this article we have developed an algorithm to decompose

(1) An Euler graphs into circuits
(2) A non-Euler graphs into circuits, paths and null graphs.

We shall recollect the definitions used in the article for a comfortable reading of the article. Decomposition of a graph G is defined as a collection of edge disjoint subgraphs $H_1, H_2, H_3 \ldots, H_k$ such that

$$\bigcup_{i=1}^{k} H_i = G, \quad \bigcap_{\substack{i,j=1 \\ i\neq j}}^{k} \big(E(H_i) \cap E(H_j)\big) = \emptyset,$$

Figure 2 presents an example of a graph G and its decomposition into 3 subgraphs H_1, H_2, H_3.

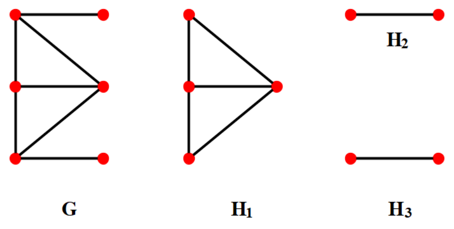


Figure 2. Example of graph G and its decomposition.

A path P_n with n vertices is an alternating sequence of vertices and edges, starting and ending with a vertex, where each vertex and edge is visited only once. A walk with n vertices is a similar sequence, but vertices can be repeated, while edges are still visited only once. The length of a path is the number of edges it contains. A closed path is called a circuit. A circuit with n vertices is denoted by C_n. A connected graph G is called Eulerian if it has a closed walk that includes all its edges of G. A graph G with no edges is called a null graph. The adjacency matrix $A = [a_{ij}]$ of a graph G with n vertices and e edges is defined as an $n \times n$ matrix such that

$$a_{ij} = \begin{cases} 1, & \text{if there is an edge between } v_i \text{ and } v_j \\ 0, & \text{otherwise} \end{cases}$$

We shall now continue to decompose G into circuits and paths.

Theorem 2.1. *Let $G = (V, E)$ be a graph such that $|V(G)| = n$. There exist subgraphs $H_1, H_2, \ldots, H_h$ of G such that*

$$\bigcup_{i=1}^{h} H_i = G, \quad \bigcap_{\substack{i,j=1 \\ i \neq j}}^{h} \Big(E(H_i) \cap E(H_j)\Big) = \emptyset,$$

such that each H_i is either a path, a circuit, or a null graph.

Proof. Let G be a graph with n vertices $\{v_1, v_2, \ldots, v_n\}$. If G is Euler, we retain the graph, else let G' be the graph obtained from G using R_3. We know that G or G' can be decomposed into circuits by R_2. Let A be the adjacency matrix of G. Let us denote every row of A as a vector. Let $r(v_i) = \langle c(a_{i1}), c(a_{i2}), \ldots, c(a_{in})\rangle$ denote the row vector corresponding to the ith row of A. By the definition of A, for every $r(v_i)$, $i = 1, 2, \ldots, n$,

$$c(a_{ij}) = \begin{cases} 1, & \text{if } v_i \text{ is adjacent to } v_j \\ 0, & \text{otherwise} \end{cases}$$

Case I Start with any row $r(v_{k_1}) = \langle c(a_{k_1 1}), c(a_{k_1 2}), \ldots, c(a_{k_1 n})\rangle$. Pick column k_2 such that $a_{k_1 k_2} \neq 0$ and $a_{k_1 k_i} = 0$, for every $i = 1, 2, \ldots, k_2 - 1$, also $k_2 \neq k_1$. This implies v_{k_1} is adjacent to v_{k_2}, but v_{k_1} is not adjacent to v_{k_i}, for $i = 1, 2, \ldots, k_2 - 1$.

Next, pick $r(v_{k_2}) = \langle c(a_{k_2 1}), c(a_{k_2 2}), \ldots, c(a_{k_2 n})\rangle$. Pick column k_3 such that $a_{k_2 k_3} \neq 0$ and $a_{k_2 k_i} = 0$, for every $i = 1, 2, \ldots, k_3 - 1$, also $k_3 \neq k_2$. This implies v_{k_2} is adjacent to v_{k_3}, but v_{k_2} is not adjacent to v_{k_i}, for $i = 1, 2, \ldots, k_3 - 1$.

Thus, v_{k_1} is adjacent to v_{k_2}, and v_{k_2} is adjacent to v_{k_3}, with k_1, k_2, k_3 distinct.

Continue this procedure to generate a sequence of vectors $v_{k_1}, v_{k_2}, \ldots, v_{k_t}$ such that v_{k_1} is adjacent to v_{k_2}, v_{k_2} is adjacent to $v_{k_3}, \ldots, v_{k_{t-1}}$ is adjacent to v_{k_t}, where $k_1, k_2, \ldots, k_t$ are distinct.

Pick row $r(v_{k_t}) = \langle c(a_{k_t 1}), c(a_{k_t 2}), \ldots, c(a_{k_t n})\rangle$. Pick column k_{t+1} such that $v_{k_{t+1}} \neq v_{k_t}$. If $v_{k_{t+1}} = v_{k_i}$ for some $i = 1, 2, \ldots, t-1$, then $v_{k_{t+1}}$ is adjacent to v_{k_i}, and v_{k_i} is adjacent to $v_{k_{i+1}}, \ldots, v_{k_{t-1}}$ is adjacent to v_{k_t}. This implies $\langle v_{k_i}, v_{k_{i+1}}, \ldots, v_{k_t}, v_{k_{t+1}}\rangle = c_{i+t+1}$.

In matrix A, set $a_{k_s k_r} = a_{k_r k_s} = 0$ for every $s = k_i, k_{i+1}, \ldots, k_t$ and $r = k_{i+1}, k_{i+2}, \ldots, k_{t+1}$. If $v_{k_{t+1}} \neq v_{k_i}$ for all $i = 1, 2, \ldots, t-1$, then continue the tracing procedure until there exists a sequence $v_{k_1}, v_{k_2}, \ldots, v_{k_t}, v_{k_{t+1}}, \ldots, v_{k_{t+p}}$, where $p \geq t+2$, such that $v_{k_{t+p}}$ is adjacent to some v_{k_i}, $i = 1, 2, \ldots, k_{t+p}-2$ to generate a sequence $v_{k_i}, v_{k_{i+1}}, \ldots, v_{k_{t+p}}$ such that $v_{k_{t+p}}$ is adjacent to v_{k_i}, v_{k_i} is adjacent to $v_{k_{i+1}}, \ldots, v_{k_{t+p-1}}$ is adjacent to $v_{k_{t+p}}$. This implies $\langle v_{k_i}, v_{k_{i+1}}, \ldots, v_{k_{t+p}} \rangle = C_{i+t+p}$.

In matrix A, set $a_{k_s k_r} = a_{k_r k_s} = 0$ for every $s = k_i, k_{i+1}, \ldots, k_{t+p-1}$ and $r = k_{i+1}, k_{i+2}, \ldots, k_{t+p}$.

Label the resulting adjacency matrix as A_1. The number of non-zero entries of matrix A_1 is at least 6 less than the number of non-zero entries of matrix A. The graph G_1 resulting from matrix A may be connected or disconnected. But G_1 is an Euler graph. In G, every vertex is of even degree. In any circuit, all vertices are of degree two. So, the vertices $v_{k_i}, v_{k_{i+1}}, \ldots, v_{k_{t+p}}$ in G_1 have degree two less than the degree of these vertices in G. Since even minus even is an even number, all vertices in G_1 have even degree, which implies G_1 is Euler. If G_1 is a disconnected graph, then each component of G_1 is an Euler graph.

Continue with G_1 to trace circuits, as we traced the circuit C_{i+t+p}. By repeating this iteration, we generate a sequence of matrices $A_1, A_2, \ldots, A_z$ such that A_z is a null matrix, and $A, A_1, A_2, \ldots, A_{z-1}$ are not null matrices. It is possible that A_z is a null matrix since by R_2, we know that G can be decomposed into circuits. Each A_i has one circuit less than A_{i-1} for every $i = 1, 2, \ldots, z$. Each iteration generates a circuit C_i, $i = 1, 2, \ldots, z-1$. By this iterative procedure, we decompose G into circuits $S_1, S_2, \ldots, S_{z-1}$.

Case II Let G be a non-Euler graph with n vertices $\{v_1, v_2, \ldots, v_n\}$. By R_3, we know that G can be modified into an Euler graph by adding edges between vertices of odd degree. Let G be non-Euler with $2k$ odd-degree vertices. Let G' be the Euler graph obtained by adding edges $E = \{e_1, e_2, \ldots, e_k\}$. Continue as discussed in Case I to generate a sequence of matrices $A, A_1, A_2, \ldots, A_z$ and hence a sequence of circuits $S_1, S_2, \ldots, S_{z-1}$.

Consider any random circuit $C_{i+t+1} = \langle v_{k_i} e_{k_i k_{i+1}} v_{k_{i+1}} \ldots v_{k_t} e_{k_t k_{t+1}} v_{k_{t+1}} \rangle$. Remove $e_{k_j k_{j+1}}$ from C_{i+t+1} for every $j = i, i+1, \ldots, t+1$, if $e_{k_j k_{j+1}} \in E$ to generate a sequence of graphs $X_1, X_2, \ldots, X_{q_1}$, where $1 \leq q_1 \leq t+i+1$. The resulting graphs X_i can be either of the following:

(1) Path $P_{k_{t+1}}$ if only one edge of $C_{i+t+1} \in E$.
(2) Either
  (a) A sequence of paths $T_1, T_2, \ldots, T_y$, where $2 \leq y \leq \left\lfloor \frac{n}{2} \right\rfloor$.
  (b) A sequence of paths $T_1, T_2, \ldots, T_x$ and a null graph N, if more than one edge of $C_{i+t+1} \in E$.
(3) A null graph N if every edge of $C_{i+t+1} \in E$.
(4) A circuit C_{i+t+1} if no edge of $C_{i+t+1} \in E$.

In general, while decomposing G into subgraphs $H_1, H_2, \ldots, H_k$, each subgraph H_i should contain at least one edge. In this case, null graphs may be generated due to the addition of edges in E to create an Euler graph. These null graphs, if any, can be neglected as they do not belong to the original graph G.

The non-Euler graph $G = G' - \{E\}$ will be decomposed into either:

(1) Circuits and paths,

(2) Circuits only,
(3) Paths only.

From Case I and Case II, we conclude that any graph G can be decomposed into subgraphs $H_1, H_2, \ldots, H_h$ such that each H_i is either a circuit or a path.

□

Remark. In any graph G, we know that there are 2k odd degree vertices. We add edges randomly between these vertices to create an Euler graph. These edges can be added in many ways. We generate a maximum number of non isomorphic Euler graphs. We note that these graphs generate different fragments. If the graphs are different the fragments generated may also be different. The graphs in Figure 3b and 3c provide two possible ways of adding edges (presented as dotted lines) to create an Euler graph for the non-Euler graph in Figure 3a. We observe that the fragments are different for these graphs as seen in Figure 3d and 3e.

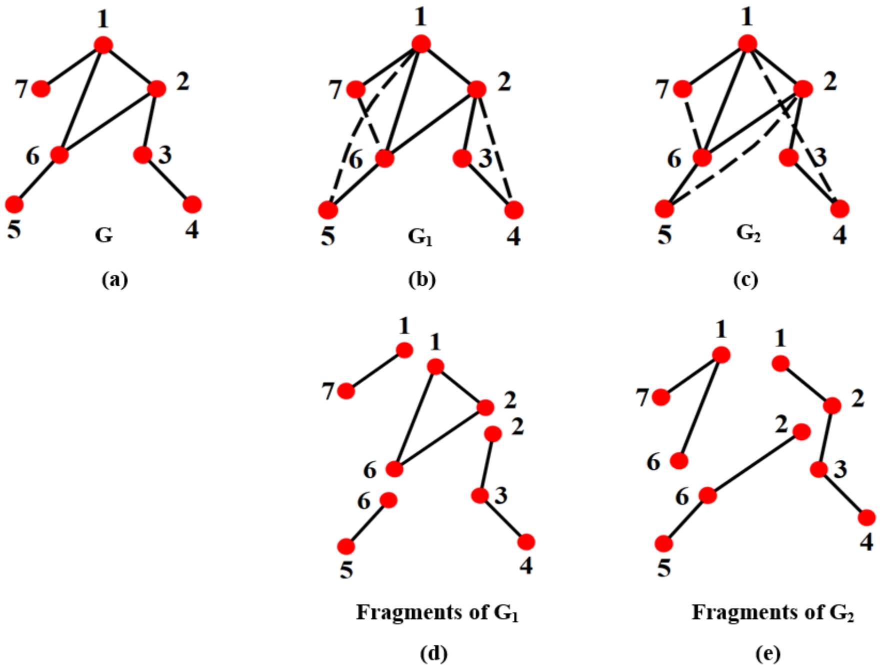


Figure 3. Fragments generated by different edge additions to G.

2.2. **Algorithm Graph Decomposition (GD).**

By Theorem 2.1 we know that G can be decomposed into circuits and paths. In this Section we continue further to develop an algorithm using Theorem 2.1. We label this algorithm as Algorithm Graph Decomposition. Algorithm GD is divided into two parts: GD1 and GD2. GD1 is designed for decomposing Euler graphs into circuits, while GD2 decomposes non-Euler graphs into circuits and paths.

GD1 GD1 requires that the vertices of the graph be assigned integers $1, 2, \ldots, n$ in any order. The algorithm identifies a circuit C_1 from the input adjacency matrix A and iteratively repeats this process to generate a sequence of circuits $C_1, C_2, \ldots, C_z$ by updating matrix A, (z – 1) times. To determine each circuit $C_i, 1 \leq i \leq z$, GD1 utilizes two primary arrays in addition to the one representing the graph. The first array is a 1-dimensional array containing a set of vertices V of an elementary path. The second array is a 2-dimensional array E containing the edges of an elementary path E. Both V and E are initially empty. As building V is a basic process, GD1 constructs the first path starting at vertex 1. This path is extended one edge at a time, subject to two conditions:

(1) The vertex to be added to the path must not be on our current list of visited vertices.
(2) The vertex to be added should not connect back to any vertex already in our path, creating a circuit.

Condition 1 ensures that an elementary path is found, while Condition 2 guarantees that each circuit is considered only once. Condition 1 is implemented in GD1 by introducing a variable J to represent the previously visited row. Initially, J is set to zero. If $V = \{v_1, v_2, \ldots v_k\}$ then to extend the path V, J is updated as v_k . This ensures that the smallest possible circuit has a length of three (as we are dealing with simple graphs), satisfying Condition 2.
At some point no vertex will be available for extension of V. A circuit confirmation C_i is now performed. i.e., there is an edge connecting the last vertex of V to any other vertex in V. In this case a circuit C_i is reported. The algorithm then continues to process the following steps.

(1) Add C_i to the circuit list C which is initially zeroed.
(2) Update matrix A by replacing non-zero entries of A to zero. Vertices included in C_i are only updated.
(3) Repeat the above process with the updated matrix starting from row 1.

Condition 2 is processed by determining the position of the already visited vertex in V for determining C_i. A substring E_1 of E is determined from the same position in E for replacing $a_{ij} = a_{ji} = 0$ for $i \neq j \in E_1$. The algorithm is terminated when the updated matrix is a null matrix. An overview of the algorithm GD1 is presented in Algorithm GD1.

Algorithm GD1:

```
**Initialize:** $S = 1$
  **While** $S \leq$ number of rows in the Adjacency matrix:
  **For** $Z_1 = 1$ to $n_1, 1 \leq n_1 \leq n$ (each element in the row):
    **If** $A(S, Z_1) = 1$:
      Initialize $r_1 = 0, J = 0, V = [\,], E = [\,], D = [\,], C = [\,]$
      **For** $r = 1$ to $n$:
        **If** $r_1 \neq 0$:
          $r = r_1$
        **end**
        For $c = 1$ to $n$:
          If $A(r,c) = 1$:
            If $c \neq J$:
              Update the values of $V$, $E$, $J$
            end
          end
          break
        end
      If $A(r,c) = 1$:
        Set $r_1 = c$
        Initialize *already visited* = 0
        For $Z_2 = 1$ to $V$:
          Check whether the next row to be travelled is *already visited*
        end
        If *already visited* = 1:
          For $Z_3 = 1$ to $E$:
            Find the position of the already travelled edge
          end
          For $Z_4 = Z_3$ to $E$:
            Replace the entries of the matrix with zero
            Update the adjacency matrix
            Store the Circuit in $D$ and $C$
          end
        break
      end
    end
    end
  Set $S = S + 1$

  end
```

GD2: Using GD1, we decompose G' into a sequence of circuits $C = C_1, C_2, \ldots, C_{z-1}$. GD2 depends on the input $E = \{e_1, e_2, \ldots, e_k\}$, an array of edges.

GD2 utilizes R_3 to generate an Euler graph G' by adding edges $e_1, e_2, \ldots, e_k$ between odd-degree vertices, as described in Theorem 2.1. GD1 is then applied to decompose G' into a sequence of circuits $C = \{C_1, C_2, \ldots, C_{z-1}\}$. GD2 operates on an input array of edges $E = \{e_1, e_2, \ldots, e_k\}$. Every e_i, $1 \leq i \leq k$, is removed from C_i, $1 \leq i \leq z-1$ by using for loops. We know that each removal generates at least one path. GD2 outputs a sequence of circuits and paths $C_1, C_2, \ldots, C_a, P_1, P_2, \ldots, P_b$, where $0 \leq a \leq z-1$ and $b \geq 1$. That is, GD2 decomposes the original non-Euler graph G into circuits and paths. An overview of GD2 is presented in Algorithm GD2.

```
**Algorithm GD2:**
**Input:** The edges to be removed
For $k = 1$ : length$(C)$:
  $C$ = circuits$\{k\}$;
  For $j = 1$ : length$(E)$:
    $E = E\{j\}$;
    For $i = 1$ : length$(C)$:
      If $C(i) = E$:
        $C(i) = []$;
        break;
      end
    end
  end
  circuits{k} = C;
end
```

Theorem 2.2. *By Algorithm GD*

(1) *All edges are exhausted.*
(2) *No circuit is repeated more than once.*
(3) *G is decomposed into cycles.*

Proof. After tracing the first circuit $\{v_1, v_2, \ldots, v_k\}, 3 \leq k \leq n$, we generate a new matrix A_1 from A by fixing $a_{12} = a_{21} = \cdots = a_{1k} = a_{k1} = 0$ in matrix A. The algorithm is then processed by initializing A_1 as A. GD terminates when the adjacency matrix A becomes a null matrix, indicating that all edges in G have been exhausted. Since the algorithm initializes a new updated matrix after tracing each circuit, no circuit is repeated. The algorithm initializes a new matrix only after completing the tracing of a circuit. The process terminates when the matrix becomes null, indicating that G has been fully decomposed into circuits, which are then printed. □

2.3. **Algorithm GD Analysis.** We have used a Lenovo Ideapad S340 equipped with a 1.6 GHz Intel Core i5-8265U processor and 12GB of 2400 MHz DDR4 RAM. MATLAB version R2023a served as the software platform for running the algorithm.

|V(G)|	T_A
1	0
2	0
3	0.026
4	0.033
5	0.041
6	0.046
7	0.05
8	0.052
9	0.053
10	0.056

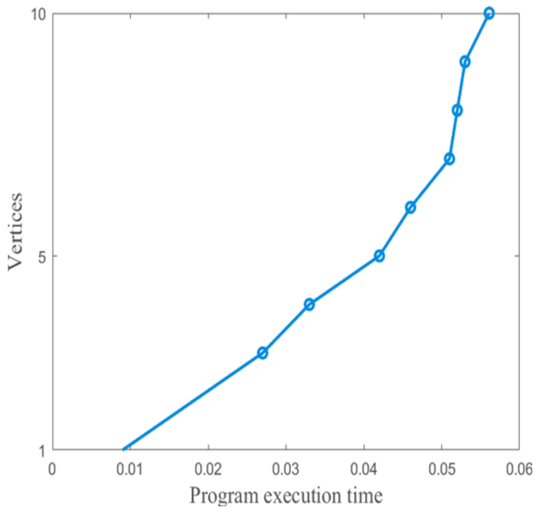


Figure 4. Runtime Estimation.

We know that an Euler graph is always connected. So, we determine the execution speed of GD using only connected graphs. The results are trivial when n = 1, 2. The number of possible circuits for a small graph is less in number. So, when n = 3, 4, 5, 6 we have estimated the execution speed for 19 random graphs. A graph that is minimally connected is a tree. Adding edges to a tree will retain back the graph connected. We start with any arbitrary tree T and estimate the run time by adding edges one at a time until we cannot continue any further. We have estimated the execution time of these 94 graphs when n = 7, 8, 9, 10. Based on these calculations, Table in Figure 4 presents the average runtime calculations. The graph in Figure 4 provides a comparison of the average execution time for n = 1 to 10. It is clear from the graph that the execution time gradually increases as n increases.

3. Algorithm GD for FBDD

Fragment based drug design has developed as an essential step in the drug discovery process. This process involves breaking down a larger drug molecule into smaller molecules, in other words into a better manageable parts or fragments that can be analysed. By doing this it is possible to focus on the key elements of a drug, often considered to contribute to the activity of drugs. These smaller fragments are also easy to analyse and manipulate which can speed up the drug discovery process. There are various methods for making this fragmentation possible like retrosynthesis analysis, manual decomposition, graph theory approaches. Graph theory approaches use molecular representation of graphs. Algorithms are developed for these graphs to identify the subgraphs (fragments) that are relevant to the drug activity. Not all fragment based drug designs target on decomposing molecules into nonoverlapping fragments. Researchers feel that nonoverlapping fragments are better since in medical fields it is important to consider the contribution of the binding energy of each of them, as different fragments are found responsible for different kinds of intermolecular interactions [17]. These are found useful in various drug discovery practice. Determining non overlapping fragments is equivalent to decomposing a chemical graph into nonoverlapping subgraphs. Algorithm GD can be used for this purpose since this algorithm decomposes a graph into paths and circuits, where no edge of G is considered more than once. We now concentrate only on nonoverlapping fragment decomposition.

Any given chemical graph can be decomposed into fragments using algorithm GD. We now further plan for using this fragmentation for drug analysis. Physiochemical properties are usually determined from the chemical structures [18]. The question now that arises is 'Can the nonoverlapping fragments be used for predicting physiochemical properties of drugs?' We now try to answer this question.

When it comes to physiochemical property prediction topological indices are in wider use. We continue this article to use topological indices and nonoverlapping drug fragmentation for physiochemical property prediction. We recollect the view of Harry Wiener which states "The boiling point of organic compounds as well as their physical properties depends functionally upon the number kind and structural arrangements of atoms in the molecules. Within the group of isomers both the number and kind of atoms are constant and variations in the physical properties are due to changes in structural interrelationships alone" [19]. He used this idea to predict the boiling point of paraffins. So, we shall choose Wiener index as a topological index for our study.

Powdery Mildew occurs when fungal growth covers the surface of plants. It often attacks foliage stems, flowers and fruits. This is noted as white powdery patches [20]. Anthracnose is a disease causing dark lesion on leaves. This often affects vegetables, flowers and fruits [21]. Black spot is a disease that affects the leaves of rose plant causing them to drop off. This reduces the plants photosynthesis which causes a decrease in rose production. Sometimes the entire plant may defoliate [22]. Apple scab is caused by a fungus that infects the leaves, buds, fruits and blossoms. This is often noticed as small spots on the surface of leaves and fruits [23]. These are few of the various fungal diseases that infect plants. Other commonly known fungal diseases are Sheath blight, Premature leaf fall, Alternaria spot/blight, Sooty blotch etc [24]. Figure 5 shows fungal infections affecting both plants and fruits.

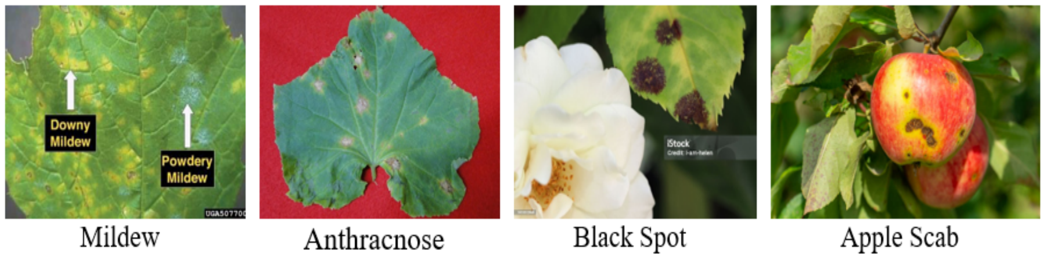


Figure 5. Various fungal diseases.

When fungicides are manufactured, care has to be taken because they may be harmful to the ecosystem. Apart from protecting the plants from infections, attention needs to be given to the surrounding environment. There are many possibilities that fungicides will move in irrigation or run off from the site of application due to storms or rainfall. This often happens because they remain in the field for many weeks. Various properties like the pesticide's field dissipation half-life, adsorption coefficient (KOC), etc., may have a negative impact on the surroundings, as fungicides last longest in the environment.

K_{OC} represents the soil adsorption coefficient, which generally depends on the hydrophobicity of the fungicides. K_{OC} determines how fungicides move in runoff. Larger K_{OC} values mean the soil absorbs the fungicides strongly. A high K_{OC} value for fungicides means that they move by associating with eroded soil or sediment particles. Lower K_{OC} values, along with higher solubility, mean that fungicides will move in dissolved form. As water solubility increases, K_{OC} generally decreases [25]. From this, we understand that pesticide applicators should have knowledge of whether the pesticide is more likely to move in dissolved form or by associating with eroded soil particles, possibly entering surface streams. These fungicides are most likely to cause toxicity to sediment organisms. Since $logK_{OC}$ values play a key role in impacting the environment, we shall focus on the $logK_{OC}$ values of fungicides.

There are various methods, such as OECD 106 and OECD 121, for the estimation of K_{OC} values [26]. OECD 121 uses HPLC for the estimation of K_{OC}, which is considered more reliable than the calculations from the QSAR method. All these methods require some basic experimental setups. If these values can be determined without such setups, it would be more user-friendly and save experimental time and costs associated with experimental implementations. We shall continue further to predict $logK_{OC}$ values of fungicides using the Wiener index and fragment-based drug design. For this purpose, we choose 10 fungicides: Azoxystrobin, Benomyl, Carpropamid, Difenoconazole, Edifenphos, Flutolanil, Hexaconazole, Iprodione, Tetrachlorophthalide, and Thiabendazole.

We shall now analyse if the Wiener index of the fragments can be as efficient as the Wiener index of the chemical graph in physiochemical property predictions. We continue as follows:

(1) Determine the chemical graph G of the fungicides.
(2) Determine the Wiener index of the graph G.
(3) Decompose G into fragments $H_1, H_2, \ldots, H_s$ using Algorithm GD to generate a graph $G_1 = H_1 \cup H_2 \cup \cdots \cup H_s$.
(4) Determine the Wiener index of G_1.

Let us now decide how to implement step 4.

3.1. **Fragment Wiener Index from Algorithm GD.**

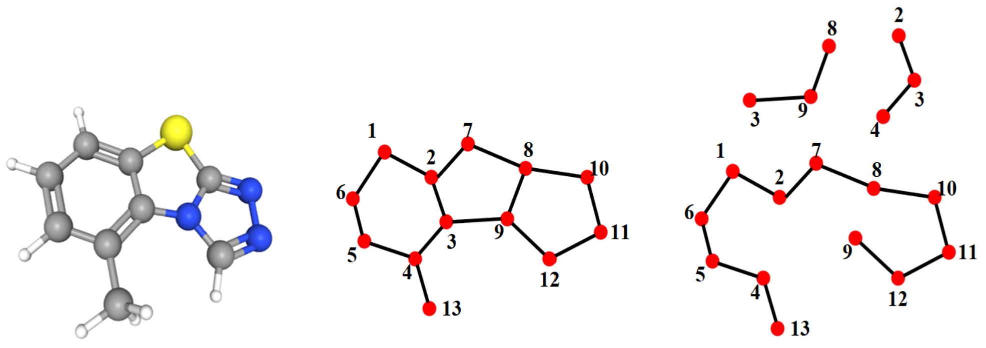


FIGURE 6. Example for Section 3.1 discussion.

Figure 6(a) presents the 3D molecular structure of Tricyclazole and Figure 6(b), the graph structure of Tricyclazole. Figure 6(c) presents the decomposed graph of the graph in Figure 6(b) using Algorithm GD by adding edges between vertices (2, 4), (3, 8) and (9, 13) respectively. The distance between vertices 9 and 13 is 3 in Figure 6(b) while it is 11 in Figure 6(c). So, we observe that when a graph is decomposed the distance between two vertices might vary drastically and hence affect the Wiener index value. To overcome this, we shall continue further to reduce the distance value. In any graph G with n vertices the length of the longest possible path between two vertices can be n – 1 and the smallest possible path between two vertices is 1. In general distance between a pair of vertices can be close to n/2 in many cases. So, whenever the length of the path of the decomposed graph is greater than n/2 we shall split the path into two paths $P_{\lfloor n/2 \rfloor}$. This will help us to maintain a value closure to the original Wiener index value. With this observation we now proceed to determine the Wiener index of the decomposed graph.

(1) Determine the chemical graph G of the fungicides from the molecular structure.
(2) Decompose G into circuits and paths $C_1, C_2, \dots, C_k, P_1, P_2, \dots, P_t$ using Algorithm GD.
(3) For any P_i, $1 \le i \le t$, if the length of P_i is greater than $\frac{n}{2}$, decompose P_i into two paths:

$$P_i = \begin{cases} P_{\lfloor n/2 \rfloor} \cup P_{\lfloor n/2 \rfloor}, & \text{if } i \text{ is odd} \\ P_{\lfloor n/2 \rfloor +1} \cup P_{\lfloor n/2 \rfloor}, & \text{if } i \text{ is even} \end{cases}$$

Applying steps 1, 2, and 3, G is decomposed into subgraphs $H_1, H_2, \dots, H_s$ where each H_i is either a circuit or a path with at most $\lfloor \frac{n}{2} \rfloor$ vertices.
(4) The Wiener index of G_1 is the sum of the Wiener indices of the subgraphs:

$$\text{Wiener index of } G_1 = \text{Wiener index of } H_1 + \text{Wiener index of } H_2 + \cdots + \text{Wiener index of } H_s.$$

For the graph in Figure 3(c), we observe that the path P_n has length greater than $\lfloor \frac{n}{2} \rfloor$. So, P_{12} is decomposed into P_7 and P_6. Thus, the final decomposition of G is $P_7 + P_6 + 2P_3$. This implies:

$$\begin{aligned}\text{Wiener index}(G) &= \text{Wiener index}(P_7) + \text{Wiener index}(P_6) + 2\times\text{Wiener index}(P_3)\\ &= 56 + 35 + (2\times 4)\\ &= 99.\end{aligned}$$

4. Results and Discussions

4.1. **Comparative Analysis.** Table 1 presents the Wiener index of G and the Wiener index of the decomposed graph G_1 and $logK_{OC}$ values for all the 10 fungicides. For $logK_{OC}$ values and molecular structures of fungicides we refer to ChemSpider [27].

S. No	Fungicides	Wiener Index of G	Wiener Index of G_1	log K_{OC}
1	Azoxystrobin	2834	122	2.869
2	Benomyl	1143	90	2.716
3	Carpropamid	828	72	4.616
4	Difenoconazole	1930	165	4.384
5	Edifenphos	730	99	2.646
6	Flutolanil	900	95	3.730
7	Hexaconazole	790	110	4.181
8	Iprodione	1024	110	2.148
9	Thiabendazole	341	66	3.345
10	Tetrachlorophthalide	258	39	2.790

Table 1. Wiener index and $logK_{OC}$ values.

The graph in Figure 7 presents the correlation between the Wiener index values of G and G_1 for all the fungicides. Figure 7 shows that there is a good correlation between the original Wiener index and the Wiener index of the fragments estimated by Algorithm GD. Also, correlation coefficient $(W, W_1) = 0.73$. Since there is a good correlation, we can use the Wiener index of the decomposed graph for $logK_{OC}$ value prediction.

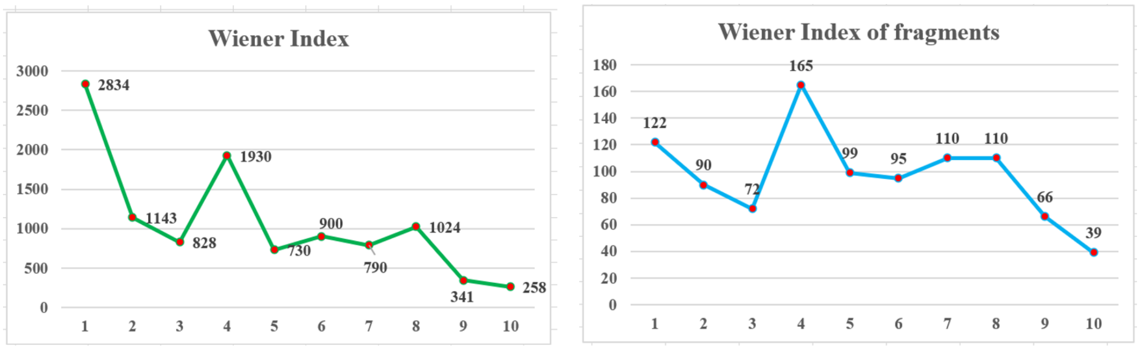


Figure 7. Correlation between Wiener index of chemical graphs and Wiener index of the fragments of these graphs.

4.2. Regression Analysis and Conclusion.

Consider the following regression equations.

$$\log K_{OC} = 3.313 + 2.709 \times 10^{-5} \times W \tag{1}$$

$$\log K_{OC} = 2.824362 + 0.005353 \times W_1 \tag{2}$$

where W is Wiener index and W_1 is Wiener index for fragments. Equation 1 presents the regression of $logK_{OC}$ with the Wiener index of the chemical graph of the 10 fungicides. Equation 2 presents the regression of $logK_{OC}$ with the Wiener index of the fragments obtained from algorithm GD for the 10 fungicides.

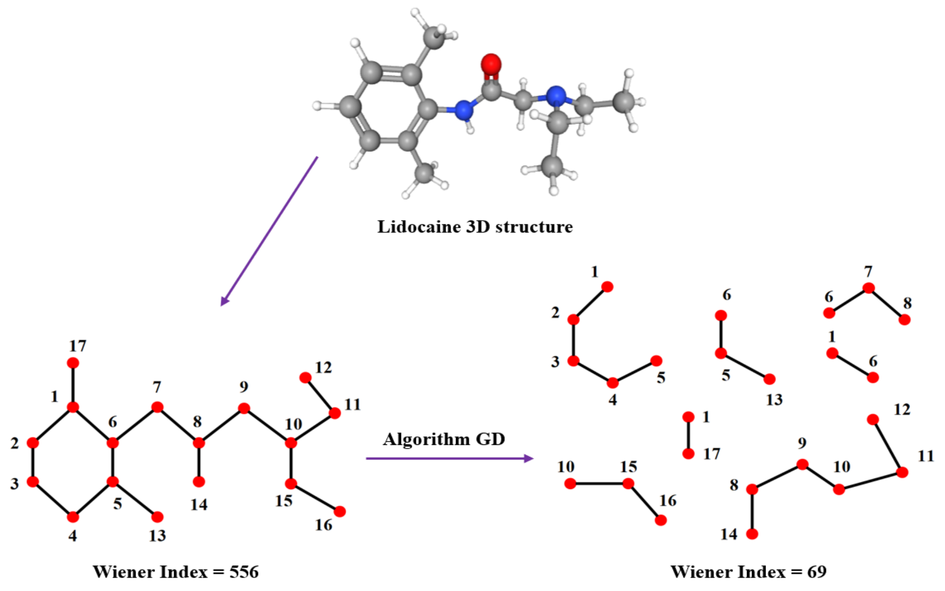
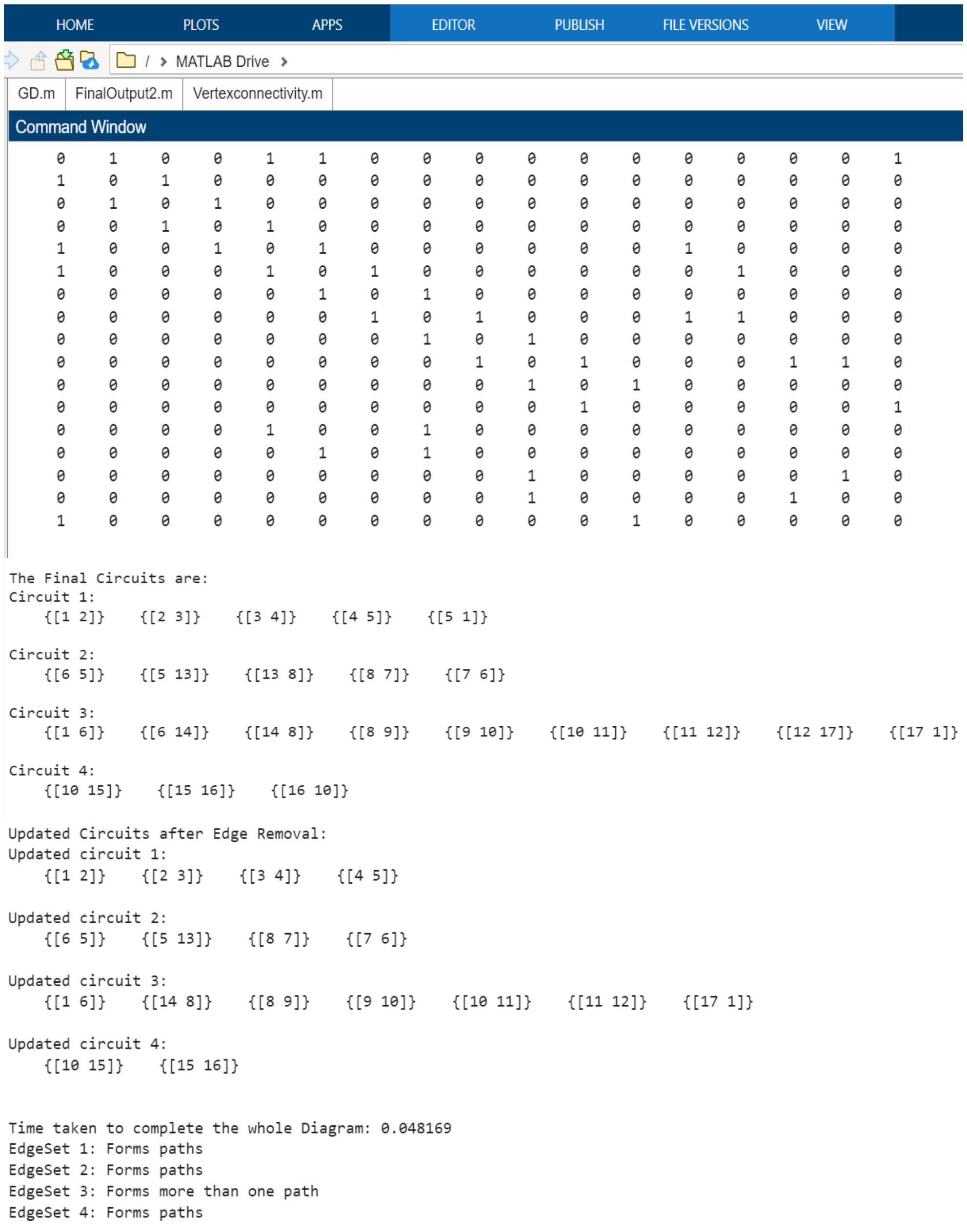


Figure 8. Wiener index Estimation for Lidocaine.

Now let us consider a new fungicide Lidocaine. Let us use Algorithm GD for decomposing the chemical graph of Lidocaine into fragments. Figure 8 presents the 3D molecular graph, chemical graph and the decomposed graph of Lidocaine using Algorithm GD structure and the corresponding Wiener index values. We observe that the chemical graph of Lidocaine is non-Euler. We add edges between the vertex pairs (1, 5), (8, 13), (12, 17), (10, 16), (6, 14), so that the resulting graph is Euler.



```
     0     1     0     0     1     1     0     0     0     0     0     0     0     0     0     0     1
     1     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     1     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     1     0     1     0     0     0     0     0     0     0     0     0     0     0     0
     1     0     0     1     0     1     0     0     0     0     0     0     1     0     0     0     0
     1     0     0     0     1     0     1     0     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     1     0     1     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     1     0     1     0     0     0     1     1     0     0     0
     0     0     0     0     0     0     0     1     0     1     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     1     0     1     0     0     0     1     1     0
     0     0     0     0     0     0     0     0     0     1     0     1     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     1
     0     0     0     0     1     0     0     1     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     0     1     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     0     0     1     0     0     0     0     1     0     0
     1     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0

The Final Circuits are:
Circuit 1:
    {[1 2]}    {[2 3]}    {[3 4]}    {[4 5]}    {[5 1]}

Circuit 2:
    {[6 5]}    {[5 13]}    {[13 8]}    {[8 7]}    {[7 6]}

Circuit 3:
    {[1 6]}    {[6 14]}    {[14 8]}    {[8 9]}    {[9 10]}    {[10 11]}    {[11 12]}    {[12 17]}    {[17 1]}

Circuit 4:
    {[10 15]}    {[15 16]}    {[16 10]}

Updated Circuits after Edge Removal:
Updated circuit 1:
    {[1 2]}    {[2 3]}    {[3 4]}    {[4 5]}

Updated circuit 2:
    {[6 5]}    {[5 13]}    {[8 7]}    {[7 6]}

Updated circuit 3:
    {[1 6]}    {[14 8]}    {[8 9]}    {[9 10]}    {[10 11]}    {[11 12]}    {[17 1]}

Updated circuit 4:
    {[10 15]}    {[15 16]}


Time taken to complete the whole Diagram: 0.048169
EdgeSet 1: Forms paths
EdgeSet 2: Forms paths
EdgeSet 3: Forms more than one path
EdgeSet 4: Forms paths
```

Figure 9. Algorithm GD output and runtime calculation for Lidocaine.

Figure 9 presents the output of the decomposed fragments for Lidocaine. Lidocaine is decomposed into 7 nonoverlapping paths. These fragments are listed out in the figure. The runtime estimation for determining the fragments of Lidocaine is also shown in the Figure 9.

Predicted $\log K_{\text{OC}}$ values from equations (1) and (2) are:

$$\begin{aligned}\log K_{OC} &= 3.313 + 0.00002709 \times 556\\ &= 3.313 + 0.01506764\\ &= 3.3281\end{aligned}$$

$$\begin{aligned}\log K_{OC} &= 2.824362 + 0.005353 \times 69\\ &= 2.824362 + 0.369357\\ &= 3.1937\end{aligned}$$

We observe that both equations give almost the same prediction of $\log K_{\text{OC}}$ values. From this discussion, we conclude that Algorithm GD can be used for determining the fragments of any chemical graph and hence can be used for predicting physiochemical properties of fungicides using the Wiener index. This method requires no experimental setup for prediction of $\log K_{\text{OC}}$ values. Thus, this approach can be used for initial prediction of physiochemical properties of fungicides, pesticides, etc.
Determining the Wiener index of graph G_1 is straightforward since Algorithm GD decomposes G into paths and circuits.
The Wiener index of these graphs is predefined:

- Wiener index of cycle C_n:

$$\text{Wiener index}(C_n) = \begin{cases} \frac{n^3}{8}, & \text{if } n \text{ is even} \\ \frac{n^3-n}{8}, & \text{if } n \text{ is odd} \end{cases}$$

- Wiener index of path P_n:

$$\text{Wiener index}(P_n) = \frac{n(n^2-1)}{6}$$

This simplifies the calculation of the Wiener index of G_1 and makes it easy to determine, requiring fewer calculations. To conclude, in this article, we have developed Algorithm GD for decomposing any graph into circuits and paths. This algorithm is used for determining the non-overlapping fragments of drugs. The fragments determined by Algorithm GD are utilized for predicting the physiochemical properties of fungicides. The proposed method does not require any experimental setup. We believe that the algorithm developed for determining non-overlapping fragments can be used for the initial prediction of physiochemical properties of drugs, and hence this algorithm can be used for decomposing small molecules for fragment-based drug design.

Conflicts of Interest: The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] B.C. Wagner, Ascending Subgraph Decompositions in Oriented Complete Balanced Tripartite Graphs, Graphs Comb. 29 (2013), 1549–1555. https://doi.org/10.1007/s00373-012-1208-5.

[2] A. Austin, B. Wagner, Ascending Subgraph Decompositions of Oriented Graphs That Factor into Triangles, Discuss. Math.: Graph Theory 42 (2020), 811–822. https://doi.org/10.7151/dmgt.2306.

[3] H. Gao, J. Wang, Y. Han, J. Sun, Enhancing Crystal Structure Prediction by Decomposition and Evolution Schemes Based on Graph Theory, Fund. Res. 1 (2021), 466–471. https://doi.org/10.1016/j.fmre.2021.06.005.

[4] J. Guo, M. Li, T. Wu, A New View toward Vertex Decomposable Graphs, Discr. Math. 345 (2022), 112953. https://doi.org/10.1016/j.disc.2022.112953.

[5] A. El-Mesady, Y.S. Hamed, H. Shabana, On the Decomposition of Circulant Graphs Using Algorithmic Approaches, Alexandria Eng. J. 61 (2022), 8263–8275. https://doi.org/10.1016/j.aej.2022.01.049.

[6] D. Saranya, S. Jeevadoss, Decomposition of Hypercube Graphs into Paths and Cycles of Length Four, AKCE Int. J. Graphs Comb. 19 (2022), 141–145. https://doi.org/10.1080/09728600.2022.2084356.

[7] K. Arthi, C. Sankari, R. Sangeetha, $\{C_n, C_4\}$-Decomposition of the Line Graph of the Complete Graph, TWMS J. Appl. Eng. Math. 12 (2022), 1441–1447.

[8] E.E.R. Merly, M. Mahiba, Some Results on Steiner Decomposition Number of Graphs, Kuwait J. Sci. 50 (2023), 1–15. https://doi.org/10.48129/kjs.16863.

[9] A.S. Issacraj, J.P. Joseph, Fork Decomposition of Some Total Graphs, Palestine J. Math. 12 (2023), 65–72.

[10] A.P. Ezhilarasi, M. Ilayaraja, A. Muthusamy, Decomposition of Tensor Product of Complete Graphs Into Cycles and Stars With Four Edges, TWMS J. Appl. Eng. Math. 13 (2023), 626–634.

[11] N.D. Tan, A Decomposition for Digraphs With Minimum Outdegree 3 Having No Vertex Disjoint Cycles of Different Lengths, Discuss. Math.: Graph Theory, 43 (2023), 573–581.

[12] Y. Xin, W. Yang, Decomposition of the Line Graph of the Complete Graph into Stars, Discr. Math. 347 (2024), 114026. https://doi.org/10.1016/j.disc.2024.114026.

[13] S. Chandran C., R. T., Common Multiples of Paths and Stars with Crowns, Proyecciones (Antofagasta) 43 (2024), 331–344. https://doi.org/10.22199/issn.0717-6279-5720.

[14] R. Campbell, F. Hörsch, B. Moore, Decompositions into Two Linear Forests of Bounded Lengths, Discr. Math. 347 (2024), 113962. https://doi.org/10.1016/j.disc.2024.113962.

[15] J.P. Gollin, K. Hendrey, S. Oum, B. Reed, Linear Bounds on Treewidth in Terms of Excluded Planar Minors, arXiv:2402.17255 [math.CO] (2024). http://arxiv.org/abs/2402.17255.

[16] L. Niu, X. Li, Decomposing Graphs of Nonnegative Characteristic into Subgraphs with Degree Restrictions, Discr. Math. 347 (2024), 113965. https://doi.org/10.1016/j.disc.2024.113965.

[17] N.N. Ivanov, D.A. Shulga, V.A. Palyulin, Decomposition of Small Molecules for Fragment-Based Drug Design, Biophysica 3 (2023), 362–372. https://doi.org/10.3390/biophysica3020024.

[18] M. Yamuna, T. Divya, Boiling Point of Alkanes and Alkenes-From Graph Eccentricity, Res. J. Pharm. Technol. 11 (2018), 1962–1970. https://doi.org/10.5958/0974-360X.2018.00364.5.

[19] H. Wiener, Structural Determination of Paraffin Boiling Points, J. Amer. Chem. Soc. 69 (1947), 17–20. https://doi.org/10.1021/ja01193a005.

[20] https://en.wikipedia.org/wiki/Powdery_mildew.

[21] https://ipm.ucanr.edu/home-and-landscape/anthracnose/pest-notes/#gsc.tab=0.

[22] https://www.istockphoto.com/photos/rose-black-spot.

[23] https://www.istockphoto.com/photos/apple-scab.

[24] https://ppqs.gov.in/divisions/cib-rc/major-uses-of-pesticides.

[25] R. Long, J. Gan, M. Nett, Pesticide Choice: Best Management Practice (BMP) for Protecting Surface Water Quality in Agriculture, University of California, Agriculture and Natural Resources, 2005.

[26] ChemSafetyPro, Soil Adsorption Coefficient (Kd/Kf/Koc/Kfoc), https://www.chemsafetypro.com/Topics/CRA/Soil_Adsorption_Coefficient_Kd_Koc.html.

[27] https://legacy.chemspider.com.