ISSN 2007-9737

Algorithm for Counting Independent Sets on Regular and Irregular Sheets of Graphene

Marlene Mijangos Romero[1,*], Cristina López Ramírez[1], Guillermo De Ita Luna[2]

[1] Universidad Juárez Autónoma de Tabasco,
División Académica de Ciencias de la Información y Tecnología,
Mexico

[2]Benemérita Universidad Autónoma de Puebla,
Facultad de Ciencias de la Computación,
Mexico

241H18003@alumno.ujat.mx, cristina.lopez@ujat.mx, deitaluna63@gmail.com

Abstract. The presence of topological defects in graphene, such as Stone-Wales configurations, significantly alters its electronic and mechanical properties. Traditional methods for counting independent sets in such structures face exponential complexity and irregular structures, limiting their applicability to nanoscale resolutions. This work introduces a novel algorithm based on Fibonacci recurrence rules, capable of handling cyclic defects through memoization and load propagation. Validated on hexagonal sets of up to 10^4 nodes, our approach reduces computation time by 15 orders of magnitude compared to brute-force methods, enabling accurate modeling of defect-engineered graphene. This advancement connects materials science with algorithmic design, providing a tool to predict stable configurations in materials.

Keywords. Graphene, topological defects, fibonacci rules, counting independent sets, complexity time.

1 Introduction

It was discovered that graphene is the world's first two-dimensional crystalline nanomaterial made of carbon atoms. It is characterized by its flexibility, strength, and resistance, as well as superhydrophobic (water-repelling) and it is superlipophilic superhydrophobic (oil-attracting) properties. This revolutionary carbon nanomaterial has the potential to be used in a wide range of applications, and holds enormous promise across various industries, including electrochemistry, optoelectronics, electronics, smart devices, aerospace, and analytical chemistry. This versatility makes its applicability relevant in materials science, energy, and biomedicine (Xuan et al., 2023). The Merrifield-Simmons index (M-S) is of utmost importance, as it is a topological indicator used in the analysis of materials in physical chemistry, particularly useful for studying electronic properties and molecular stability, which is required for research on graphene (Merrifield & Simmons, 1989).

Ones of the most recent studies on independent sets is (De Ita, Bello & Contreras, 2023), where an innovative algorithm was created to count independent sets within grid-like structures to compute the number of independent sets for a square grid formed by m rows and n columns. In Figure 1, we can visualize a regular graphene sheet, free of defects.

Studies have been conducted, such as (Goft et al., 2023), which discusses hexagonal graphene lattices. The problem of counting independent sets in graphs has been widely addressed in the literature due to its relevance in computational chemistry, network theory, and molecular structures like benzenoids. Various strategies have been proposed in recent years to tackle this problem, each with different trade-offs between accuracy and computational efficiency.

Among the most representative methods are:

- Transfer Matrix Method: Traditionally used in statistical physics and theoretical chemistry.

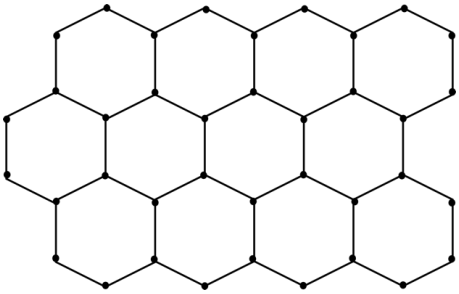


Fig. 1. Regular graphene mesh

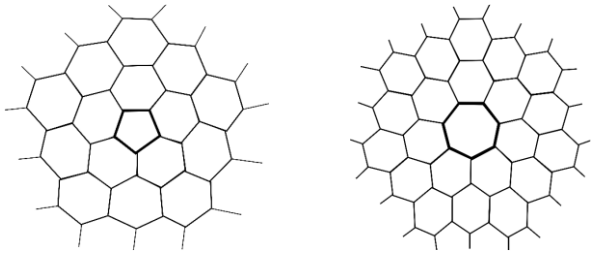


Fig. 2. Irregular mesh with a) pentagon b) heptagon

This method accurately calculates the number of independent sets in regular grid graphs. It constructs a symmetric matrix whose entries encode valid column configurations—those without consecutive ones—representing possible arrangements of vertices in independent sets. The graph's structure and evolution are modeled by iterating this matrix, systematically incorporating local adjacency constraints between consecutive columns (Calkin & Wilf, 1998).

- The Merrifield-Simmons vector is a mathematical tool defined to represent paths within chemical structures known as double hexagonal chains, also called benzenoids. This vector accurately captures the structural characteristics of these compounds, which are fundamental in chemical graph theory (Oz & Cangul, 2021).

Despite the utility of these methods, their applicability is limited in irregular structures or those with cyclic defects, or what are also called topological defects, such as the Stone-Wales (S-W) defect, where molecular structures like pentagons can induce geometric deformation and electronic perturbations, affecting the photophysical properties of the material (Fei & Liu, 2022). Likewise, heptagons play a fundamental role in understanding the electronic properties of graphene, as their presence can modify the binding energy peaks observed in X-ray photoelectron spectroscopy (XPS) analyses (Kim et al., 2021). We present this type of structure in figure 2.

Figure 2 present irregular graphene sets, and we observe the "Stone-Wales" defects in these sets (Goft et al., 2023). Pentagons and heptagons can also be represented together, and there can be more than one. Currently, efficient algorithms for irregular sets with these types of cycles (pentagons/heptagons) are lacking. To work with these defective graphenes, new and advanced procedures are needed.

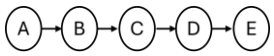
In our algorithmic proposal, we introduce a new way to attack asymmetries of any irregular graphene, we consider a Hamiltonian path (Hw) to traverse by the graphene sheet. With our new algorithm, we can address these defects by applying Fibonacci rules, subtraction, and product rules using memoization, which significantly reduces the time complexity of the algorithms for counting independent sets.

2 Notation

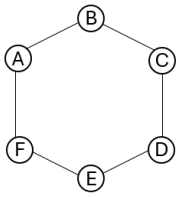
We consider $G = (V, E)$ as an undirected graph, where V is the set of vertices and E is the set of edges. We assume that G is a simple graph, which implies that it contains no loops or multiple (or parallel) edges. The edge connecting vertices u and *v* is denoted as uv; sometimes $\{u, v\}$ is used to denote the edge uv, in which case it is said that vertices *u* and *v* are adjacent.

The degree of the graph G is $\Delta(G) = max\{\delta(x): x \in V\}$, where $\delta(x)$ is the degree of the vertex x . We denote: Simple path $P_{n\text{-}1}$ as a path where all vertices are distinct, as shown in Figure 3 (a). A simple cycle (Cn) is a path in which the first and last vertex coincide (Figure 3 (b)).A Tree (Tn) is a graph without cycles and connected, where there are *n* vertices and $n - 1$ edges, as it is shown in Figure 3 (c).

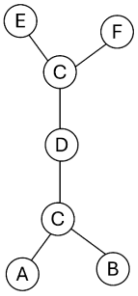
For a graph $G = (V, E)$, a set S is independent if, for every pair of vertices x and y in *S*, there is no edge connecting them. $I(G)$ represents the set of



a) Simple path



b) Simple cycle



c) Tree graphs

Fig. 3. Simple topologies; a) Simple path, b) Simple cycle, c) Tree graphs

all independent sets of G, while $I_{-v}(G)$ ($I_v(G)$) are all independent sets where the vertex v does not appear (appear). Meanwhile, $i(G) = |I(G)|$ is the number of total independent sets in G.

Calculating $i(G)$ is an NP-complete problem (Any problem in the NP class can be transformed or reduced to this problem in polynomial time) (Greenhill, 2000) for graphs G, where $\Delta(G) \geq 3$. Even when restricted to 3-regular graphs, the calculation of $i(G)$ remains NP- complete (Cormen et al., 2009).

In (Calkin & Wilf, 1998), the authors introduce the transfer matrix method for computing $i(G)$, considering the graph as a grid graph $G_{m,n}$. However, this method has an explosive exponential time complexity. Contrary, we will apply a set of basic counting rules for computing $i(G)$, and considering first, simple topologies as simple cycles, trees, or a regular graphene structure.

We consider that counting the number of independent sets of a graph G involves traversing its Hamiltonian path H_c as described in (El-Basil, 1988). Each vertex $v \in V(G)$ is associated with a pair (α_v, β_v), called its load, where $\alpha_v = |I_{-v}(G)|$ and $\beta_v = |I_v(G)|$. In our method, the load of a vertex $v \in V(G)$ is calculated at the same time that v is visited during a traversal in G. Therefore, the load of v is an auxiliary and temporary pair used to determine the number of independent sets of G, such that if v_r is the last vertex visited during the traversal in G, then $i(G) = \alpha_{vr} + \beta_{vr}$. In the case of a path, the Fibonacci rule (1) is applied as:

$$\alpha_{i+1} = \alpha_i + \beta_{i;} \qquad \beta_{i+1} = \alpha_i. \tag{1}$$

The sequence (α_i, β_i), for i = 1,..., n+1, constructed from recurrence (1), allows us to obtain $i(P_j) = \alpha_j + \beta_j$, for j = 1,...,$n+1$ for any path of $n+1$ vertices. Therefore, the calculation of $i(G)$ is based on the incremental computation of $i(P_j)$, for j = 1,...,n + 1. We use the symbol → to denote the application of recurrence (1) on (α_i, β_i) in order to obtain (α_{i+1}, β_{i+1}), such recurrence (1) is called the *Fibonacci rule*.

For example, by walking by a path P_n in a linear and incremental way, we obtain for (α_{n+1}, β_{n+1}) the pair (F_{n+2}, F_{n+1}), where F_n is the *nth*- Fibonacci number (El-Basil, 1988), (Oz & Cangul, 2021). Then, $i(P_n) = F_{n+2} + F_{n+1} = F_{n+3}$. For example, the computation of $i(P_5)$ is given as: (1, 1) → (2, 1) → (3, 2) → (5, 3) → (8, 5) → (13, 8), and $i(P_5) = 13 + 8 = 21 = F_8$.

The sequence (α_i, β_i), for i = 1,...,n (generated during the calculation of $i(P_{n-1})$ is referred to as a computation thread (or simply a thread). It is important to note that each temporal load (α_i, β_i) can be stored in a structure associated with each vertex v_i. For the computation of $i(G)$, we build a Hamiltonian path (that is, a route that visits each vertex exactly once). Our approach classifies each edge as a tree edge or as a frond edge, which facilitates the incremental calculation of the number of independent sets.

3 Processing Frond Edges in Trees and Benzenoid Graphs

Let $T(V_r)$ be a tree with its root at vertex V_r (Cormen et al., 2009). In T, the vertices of degree one are called leaves or dangling nodes, while the internal vertices are those with a degree greater than one,

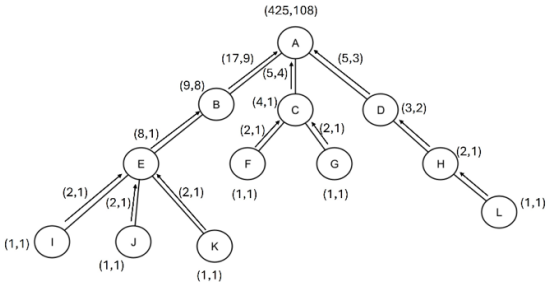


Fig. 4. Computing the M-S index on a tree

excluding the root. Figure 4 illustrates this structure. To analyze the tree, we apply a post-order traversal, associating each vertex $v_i \in V$ with its load (α_i, β_i), defined as: $\alpha_i = |I_{-vi}(T_i)|, \beta_i = |I_{vi}(T_i)|$, where T_i is the induced sub-tree $T/V\ (T_i)$. Thus, the total number of independent sets in T_i is obtained as: $i(T_i) = \alpha_i + \beta_i$.

For each leaf node $v \in T$, the associated load is $(\alpha_v, \beta_v) = (1,1)$ which is shown at the vertices $I,\ J,\ K,\ F,\ G\ y\ L$ in Figure 4. This is because for any induced leaf *v* of the tree, $I(T_v) = \{\emptyset, \{v\}\}$. The Pair $(\alpha_{vi+1}, \beta_{vi+1})$ for a vertex in the traversal is constructed from the previous pair $(\alpha_{vi}, \beta_{vi})$ using the Fibonacci recurrence (1). This recurrence is applied to the edges $I-E,\ J-E,\ K-E,\ F-C,\ G-C,\ L-H$, as seen in Figure 4. Since the node H has no children, the process continues uninterrupted to nodes D and subsequently A.

When a node v_i in the tree T has multiple children, meaning v_i has k children, the accumulation of information in the parent node is modeled using the Hadamard product rule (2) over the pairs $(\alpha_{ij},\ \beta_{ij})$, with *j*=1,…,*k*, to obtain $(\alpha_i,\ \beta_i)$.

For example, if $(\alpha_{i1},\ \beta_{i1})$ and (α_{i2}, β_{i2}) are the loads of the two children v_{i1} and v_{i2}, with a parent v_i, they are updated as follows:

$$(\alpha_i, \beta_i) = (\alpha_{i1} \cdot \alpha_{i2}, \beta_{i1.} \cdot \beta_{i2}). \quad (2)$$

This process is illustrated in Figure 4, where node *E* has three children. The application of the Fibonacci recurrence for each child produces the pairs $(2,1), (2,1)\ y\ (2,1)$, which, when combined using the Hadamard product rule (2), results in: $(\alpha_E, \beta_E) = (2 \cdot 2 \cdot 2, 1 \cdot 1 \cdot 1) = (8,1)$. Moving up the hierarchy, the Fibonacci recurrence is applied again to obtain: $(\alpha_B,\ \beta_B) = (8+1, 8) = (9,8)$. Repeating this procedure up to the root yields: $(\alpha_A, \beta_A) = (9+8, 9) = (17,9)$. Finally, at the root v_r, where nodes B, C, and D converge, the product rule is applied again: $(\alpha_A, \beta_A) = (425,108)$, and $i(T) = 425 + 108 = 533$.

In our algorithmic proposal, frond edges are processed in two phases. To illustrate this procedure, let us consider the tree T represented in Figure 5, where one of its nodes is connected to a benzenoid (Doslic & Zubac, 2018), which in turn is linked to another benzenoid. The traversal of T is done in post-order. We begin processing at vertex *L*, located at the rightmost end of the tree.

Phase 1: Creation of Threads in the Frond Edge. In the first phase of processing a frond edge $\{u, v\}$, let us assume that $(\alpha_v, \beta_v)i$ is the pair associated with the active thread L_i. At the time vertex *v* is visited, a new subordinate thread L_{vwi} is generated, which directly depends on the master thread L_i. Its initial load is defined as: $(\alpha_{vw}, \beta_{vw})_{vwi} = (0, \beta_v)_{vwi}$.

This procedure is applied to all active threads L_i where $\beta_v > 0$. In this way, the subordinate thread L_{vwi} inherits a label v_{wi} that acts as a pointer to its master thread L_i.

In Figure 5, while processing vertex L, both the primary thread L_p and the secondary thread L_s are created, resulting in the initial assignments: $L_p = (1,1)$ and $L_s = (0,1)$.

Phase 2: Application of the Subtraction Rule. The second phase of processing the frond edge $\{v, w\}$ occurs when the traversal reaches vertex w, having previously marked *v* as a visited vertex. Under this circumstance, control of the traversal remains at *w*, preventing redundant visits within the graph.

Let us consider $(\alpha_w,\ \beta_w)_i$ and $(\alpha_{vw},\ \beta_{vw})_{vwi}$ the loads in *w* associated with the master thread L_i and the subordinate thread L_{vwi} respectively. The subtraction rule (3) allows us to update the load of *w* in L_i:

$$(\alpha_w\ ,\ \beta_{wi}\ )_{\ i} = (\alpha_w\ , \beta_w - \beta_{vw})_i. \quad (3)$$

After applying this rule, all subordinate threads L_{vwi} are closed, thereby reducing the number of active threads. This procedure is exemplified in Figure 5, where in the first cycle, the traversal moves from vertex P to K, at which point the application of the Fibonacci recurrence is completed. At this stage, the cycle closes by

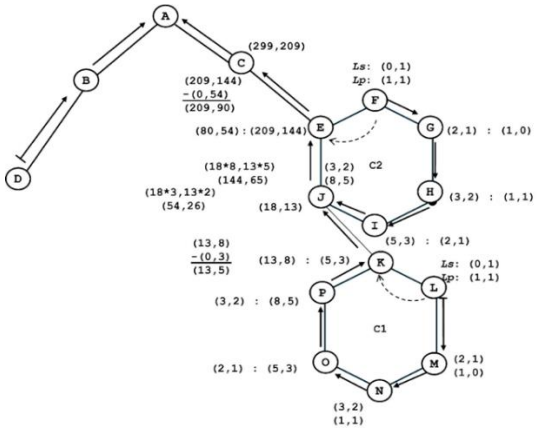


Fig. 5. Tree traversal with hexagonal graphs

applying the subtraction rule: $(13, 8-3) \Rightarrow (13, 5)$. This reduction in the number of active threads enables the traversal to continue efficiently. Moving from vertex K to vertex J, we again apply the Fibonacci rule, obtaining: (18,13) and store this value at vertex J.

For the second cycle, the process begins at vertex F, applying the Fibonacci rule: $L_p = (1,1)$, $L_s = (0,1)$. Upon reaching vertex J, where a stored value already exists, the Hadamard product is applied: $L_p = (18 \cdot 18, 13 \cdot 5) = (144,65)$ and $L_s = (18 \cdot 3,13 \cdot 2) = (54,26)$. Subsequently, as we move toward vertex E, we again apply the Fibonacci recurrence: $L_p = (209,144)$, $L_s = (80, 54)$. At this point, we encounter a frond edge, prompting us to apply the subtraction rule: $(209,144 - 54) = (209,90)$. Continuing our journey to vertex C, the values update to: $L_p = (299,209)$.

These values, depicted in Figure 5, illustrate the sequential application of the rules according to the structural topology of the graph. This method proves to be particularly useful in our algorithmic proposal for the representation and analysis of regular graphene sheets.

4 Processing Regular Graphene Sheets

A graphene sheet is formed by carbon atoms by sp^2 bonds that are organized in a hexagonal lattice structure (Xuan et al., 2023). Let us denote by $H_{r,t}$ a regular graphene sheet (GS), with r rows and t columns of hexagons. In $H_{r,t}$ all faces, except for the outer ones, are hexagonal in shape, and it contains a total of $r \cdot t$ hexagons. Each row of this structure is offset by half a hexagon relative to the adjacent row, as illustrated in Figure 6. This arrangement allows two hexagons to share an edge or remain independent. Additionally, $H_{r,t}$ can also be interpreted as the molecular graph of a regular Benzenoid system.

Benzenoid systems (BS) are molecular structures with notable geometric characteristics. They appear as connected and infinite chains of benzene molecules, where each pair of adjacent benzenes shares a single common edge. These structures are built following a specific rule, taking a benzene molecule as the fundamental unit. Benzenoid systems are highly relevant in theoretical chemistry, as they constitute the quintessential graphical representation of benzenoid hydrocarbons (Kwun et al., 2018).

The vertices lying on the border of the non-hexagonal face of a BS are called external, other vertices, if any, are called internal. A BS graph without internal vertices is called catacondensed. Pericondensed BS has internal vertices (Gutman, 2018).

In this section, we present a method to calculate $i(H_{r,t})$, where $H_{r,t}$ represents a regular graphene sheet. Unlike the transfer matrix method (El-Basil, 1988), our proposal does not require an exponential number of basic operations. This method takes advantage of the existing symmetry between two contiguous rows of $H_{r,t}$, allowing us to construct a single Hamiltonian path (Hw) organized by rows. This path Hw will visit each vertex exactly once. Additionally, each edge of $H_{r,t}$ will be classified as either a tree edge or a frond edge. The computations can be stored in a table $T_{k,l}$, with *k* rows and *l* columns, to store the partial calculations for each vertex visited through the traversing of the Hamiltonian path on $H_{r,t}$.

Except for the first and last vertices visited by the Hamiltonian path, the two edges of the vertices with degree 2 are considered tree edges and they are processed using the Fibonacci rule. On the other hand, at the vertices of degree 3, two of their edges are visited as tree edges by Hw, while the third one is classified as a frond edge and is processed using the subtraction rule.

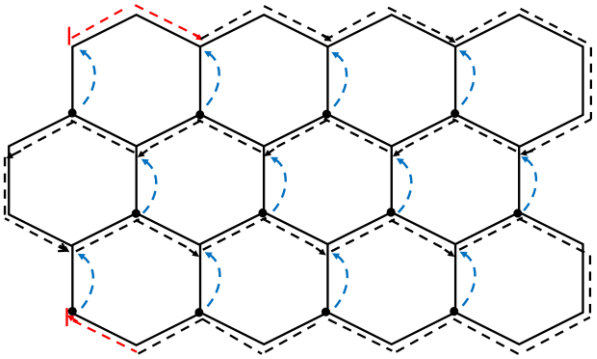


Fig. 6. Graphene mesh

During the traversal in $H_{r,t}$, when a vertex v is visited that marks the start of a frond edge $\{v,w\}$, as many new threads are generated as there are active lines in $T_{k,l}$, with a value of $\beta_v > 0$.These new threads will remain open until Hw visits vertex w. At that moment, the threads are closed, and it is possible to reorganize the rows in $T_{k,l}$ to retain only the active threads. When some rows in $T_{k,l}$ are moved, it will be necessary to update the pointers used in the thread labels. However, the value of k is adjusted to reflect the maximum number of active threads required at any moment during the calculation of $i(H_{r,t})$.The number of rows k on $T_{k,l}$ is mainly determined by the maximum number of active threads during the whole computation of $i(H_{r,t})$. The value k is dynamic and varies according to the number of frond edges in which only one of its vertices has been visited by Hw.

When only vertex v has been traversed by Hw in a frond edge $\{v,w\}$, it is said that Hw has an open cycle, which will close when Hw visits vertex w. Thus, k represents the maximum number of active open cycles during the traversal of Hw. On the other hand, the number of columns l in the table $T_{k,l}$ corresponds to the total number of visited vertices, that is, $l = |V(H_{r,t})|$.

The traversal Hw can be interpreted as the application of a DFS (Depth-First Search) algorithm on the graph $H_{r,t}$, where the diagonal edges are recognized as tree edges and each frond edge is associated with a vertical edge of $H_{r,t}$. The dfs traverses all vertices in row 1 until it reaches vertex $(1,n)$. During this traversal, the vertices $(1,j)$ with an adjacent vertex $(2,j+1)$, *where* $j = 1,\ldots,n-1$ indicate the start of an cycle.

After traversing the first row of $H_{r,t}$, the path moves from $(1,n)$ to $(2,\ n+1)$, and at the same time, a Fibonacci recurrence is applied to the load pairs of the active threads. Next, the traversal visits all vertices in row 2 with diagonal and leftward movements, that is, from $(2,j)$ to $(2,j-1)$, where $j = n+1,\ldots,2$, until it reaches vertex (2,1).

Similar to the first row, the vertices $(2,j)$ with a neighboring vertex $(3,j)$,where $j = 1,\ldots,n$ mark the beginning of a cycle. It is important to note that the vertical edge $\{(2,2),(1,1)\}$ is considered a frond edge in this traversal.

However, starting from the second row onward, there are also vertices that are terminal ends of frond edges. During the traversal in $H_{r,t}$, these ends are recognized and processed in $T_{k,l}$, although they do not affect the vertex being visited at that moment.

From vertex $(2,1)$, the traversal moves to vertex (3,1) and then visits all vertices in row 3 of $H_{r,t}$ moving to the right, from $(3,j)$ to $(3,j+1)$, where $j = 1,\ldots,n-1$. Again, when the traversal encounters a vertex with incident vertical edges, these edges are processed as the start of new cycles, the closure of open cycles, or as a path movement to change rows (in the case of edges of the type $\{(3,j),(2,j)\}$, where $j = 3,\ldots,n$.

In general, diagonal edges are processed using the Fibonacci recurrence (Equation 1). Meanwhile, vertical edges indicate the start of new cycles, the closure of open cycles, or a row change in the traversal. When the terminal ends of frond edges are visited, the subtraction rule (Equation 3) is applied.

In this way, the path visits all the vertices in $H_{r,t}$ with only diagonal movements in the same row, while keeping movements to the right for odd rows, and movements to the left for even rows of $H_{r,t}$. Also, the path only performs one vertical movement when it changes row.

5 Processing Irregular Graphene Sheets

Usually, the Hamiltonian path (Hw) on a regular GS requires only one starting point. However, in order

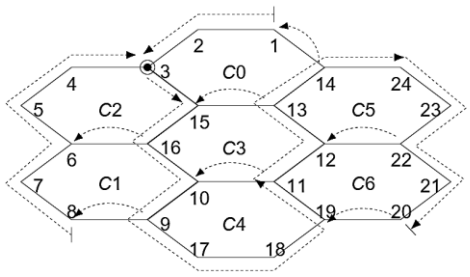


Fig. 7. Hamiltonian path on an irregular benzenoid

Algorithm 1. Procedure CountndependentSets(G)

```
Procedure CountIndepe ndentSets(G):
G_immutable←ConvertToImmutableRepresentation(G)
      Return
CountRec(G_immutable)
end Procedure
```

Table 1. The number of independent sets of vertices 1-3 corresponding to Figure 7

	1		**2**		**3**
L_p	(1,1)	→	(2, 1)	→	(3,2)
C_0	(0,1)	→	(1,0)	→	(1,1)

Table 2. The number of independent sets of vertices 8-3 corresponding to Figure 7

	8		**7**		**6**		**5**		**4**		**3**
L_p	(1,1)	→	(2,1)	→	(3,2)	→	(5,3)	→	(8,5)	→	(13,8)
C_1	(0,1)	→	(1,0)	→	(1,1)	→	(2,1)	→	(3,2)	→	(5,3)
C_2 L_p					(0,2)	→	(2,0)	→	(2,2)	→	(4,2)
C_2 C_1					(0,1)	→	(1,0)	→	(1,1)	→	(2,1)

to compute the M-S index on an irregular benzenoid system (IGS), it is possible to require different starting points for the Hamiltonian path. This leads to the formation of various starting points for Hw over IGS.

For example, let us consider the irregular IGS from Figure 7. For this graphene sheet (IGS), it is necessary to use different starting points to build a Hamiltonian path on IGS.

It is possible to have as many starting points as you need, in order to build a Hamiltonian path (Hw) over IGS. The purpose is that Hw allows us not only to visit every vertex, but also to access any edge of IGS. As the Hw for BS, there are tree edges and frond edges to be processed. The vertices of degree 3 in IGS have two tree edges and one frond edge.

When a Hamiltonian path (Hw) on IGS is built, it is relevant to identify the extreme hexagons in each row of IGS. Since in the worst case, a new search line of the Hw has to be formed per each row of IGS . Each starting point of Hw would have associated two computing threads, since one of the edges incident to the starting vertex will be a frond edge. And, each starting point of Hw comes from a vertex of degree 2 as well as it has been part of one of the extremal hexagons of a row. If there are no vertices of degree 2 in a row of IGS, it implies that every vertex of that row can be visited through the search line of the previous row of hexagons.

Algorithm 2. Function Recursive Independent Set Counting

```
Function CountRec (G_frozen):

Input: G_frozen {An immutable graph representation}
Output: Total number of independent sets in G_frozen
  If G_frozen is empty Then
    Return 1 {Base case: An empty graph has one independent set (the empty set)}
  end if
  if IsPath(G_frozen)Then
    n ← Number of nodes in G_frozen
    Return  Fibonacci(n + 2) {Shortcut: Use Fibonacci for linear paths}
  end iIf
  Components ← FindConnectedComponents(G_frozen)
  if |Components| > 1 Then
     Result ← 1
    for each Component in Components Do
                SubG ←Subgraph induced by Component in G_frozen
                Result ← Result * CountRec(SubG)
    end for
    Return  Result {Product of results across components}
  end if
   v ← Vertex with minimum degree in G_frozen
  Neighbors ← Set of neighbors of v in G_frozen
  G_without_v ← Subgraph obtained by removing v from G_frozen
  G_without_v_and_Neighbors ← Subgraph obtained by removing v and its neighbors    from  G_frozen
  Return CountRec(G_without_v) + CountRec(G_without_v_and_Neighbors)
end Function
```

Algorithm 3. Procedure Finding Connected components

```
Procedure FindConnectedComponents(G):

Input: G{Graph represented as (V, E)}
Output: List of connected subgraphs (components) in G
  Visited ← Empty set
  Components ← Empty list
  for each v in V(G) do
    if v is not in Visited Then
      Component ← Empty Subgraph
      Queue ← [v] {Initialize queue for traversal}
      while Queue is not empty do
        u ← Queue.pop()
        if u is not in Visited Then
          Visited ← Visited ∪ {u}
          Add u and its neighbors to Component
                        Queue.extend(all unvisited neighbors of u in G)
        end If
      end while
      Append Component to Components
    end if
  end for
  Return Components
end Procedure
```

Algorithm 4. Function Path Detection

```
Function IsPath(G):
Input: G {Graph represented as(V, E)}
Output: Boolean indicating if G is a linear path graph
  Degrees ← List of degrees for each node in V(G)
    NumDegree1 ← Count of nodes with degree = 1
  Return (All degrees ≤ 2) AND (NumDegree1 == 2)
end Function
```

Algorithm 5. Function: Fibonacci Sequence

```
Function Fibonacci(n):
Input: n {Position in the Fibonacci sequence}
Output: n-th Fibonacci number
  if n == 0 Then
    Return 0
  end if
  if n == 1 Then
    Return 1
  end if
  a ← 0
  b ← 1
  for I ← 2 to n Do
    Temp ← b
    b ← a + b
    a ← Temp
  end for
  Return b
end Function
```

Algorithm 6. Procedure Immutable Graph Representation

```
Procedure ConvertToImmutableRepresentation(G):
Input: G {Graph represented as (V, E)}
Output: Immutable representation of G
  Return {Each vertex and its neighbors represented as frozensets}
End Procedure
```

Two contiguous search lines of Hw, that is, the lines $r(i)$ and $r(i+1)$ associated to the vertices-row i and $i+1$ of IGS, both have opposite directions in Hw. Therefore, one of them visits vertices from left to right (or from top to down), and the other line visits vertices from right to left (or from down to top), so that both search lines of Hw will meet at a common vertex u. We draw the meeting of different search lines with the symbol $\odot$.

For example, when the M-S index on the IGS from Figure 7 is calculated, two threads are created beginning at vertex 1. Those threads are denoted by L_p and C_0. Afterward, the search line of Hw will visit vertices 2 and 3.

The other search line for Hw starts from vertex 8. Afterward, this search line of Hw will visit vertices: $8-7-6-5-4-3$. As two hexagonal cycles (C_1 and C_2) are part of those vertices, then four computing threads are formed.

The threads for both search lines of Hw converge at vertex 3, then vertex 3 is a meet vertex of Hw. Both search lines merge into a single search line in the vertex 3. As the product rule has to be

applied for the meet vertices, then a multiplicative process among the threads of the two different paths is formed.

In general, if we assume that Hw has a search line $r(i)$ with $l(i)$ computing threads associated, and there is another search line of Hw $r(i+1)$ with $l(i+1)$ computing threads associated. Then, when both search lines converge in a meet vertex u, both search lines merge into a single search line, as it happens when the search line of two leaf vertices converge in the parent node of both leaves. The Hadamard product is applied between each one of the pairs of the threads of $l(i)$ against each one of the pairs of the computation threads of *l*(*i*+1). Thus, a total of $l(i) \cdot l(i+1)$ Hadamard products are performed.

Note that the graphene sheet could contain stone-wales, and our algorithm continues working for these cases, since the irregularities in the sheet; given by the number of sides of the polygon, as well as the number of them per row, it does not change the existence of a Hamiltonian path on the system.

In this way, we propose a systematic and general method based on the construction of a Hamiltonian path that allows us to recognize each edge in IGS as a tree edge or as a frond edge. In turn, this allows us to compute the M-S index on any IGS. With purposes of illustration, we present a Hamiltonian path by rows for the IGS, in Figure 7, and its corresponding M-S index computation in Table 1 and 2. The Hamiltonian path over an IGS has a linear-time complexity on the number of edges in IGS. Meanwhile, the time-complexity of the computation for obtaining the M-S index of IGS depends on the maximum number of computation threads that remain active at any time of the path.

Let $t(i)$ be the number of hexagons in the row i of IGS. And let $t = max\{t(i): i = 1, \ldots, m\}$ be the maximum number of hexagons for any row of IGS.

There is a frond edge for each hexagon in IGS, so we have a maximum of t frond edges per row. The number of computing threads that must be active at any time is a power of 2 over the number of frond edges that can be pending for performing the second step of the frond edge processes. This results in a number of $O(2^{t+1})$ active computation threads at any time during the computation of $i(IGS)$.

Therefore, we have that our proposal for the computation of the M-S index on any IGS is deterministic and exact, and it has a time-complexity of order $O(2^{t+1} \cdot |E|)$, where$|E|$ is the number of edges in the IGS and t is the maximum number of hexagons in any row of IGS.

Notice that the Hamiltonian path on IGS is not unique, since different Hamiltonian paths could be built on an IGS. However, in order to build a minimum number of computing threads, and thus, to reduce the running time for the computation of $i(IGS)$, it is better to build a Hamiltonian path with a minimum number of starting points.

The non-regularity of the number of hexagons per row in IGS does not permit the application of the transfer matrix method, as well as the application of the M-S vector for computing the M-S index on IGS (Oz & Cangul, 2021), (Oz & Cangul, 2022). Instead, our proposal can be applied on regular or irregular sheets of graphene. Furthermore, the time-complexity of our method has an exponential behavior only on one of the dimensions of such graphene system.

The following shows the procedures and functions of the Algorithm for Counting Independent Sets on Regular and Irregular Sheets of Graphene.

6 Comparison and Validation of the Proposed Algorithm

To evaluate the applicability and efficiency of different approaches for counting independent sets in graphene-like molecular structures, a comparative table was constructed between three widely used methods: (i) the recursive algorithm proposed in this work optimized for irregular graphene sheets (IGS), (ii) the classical transfer matrix method, commonly applied to systems with regular or repetitive structures, and (iii) traditional approaches used in benzenoid systems, such as the Merrifield-Simmons (M-S) vector and combinatorial techniques based on structural symmetry. The algorithm developed in this research is designed to operate on irregular structures, where the number of hexagons per row can vary. Unlike the transfer matrix method, which requires segmentation of the graph into homogeneous columns to iterate through matrix

Table 3. Key differences regarding structural requirements, graph coverage, computational complexity, and applicability

Method/Approach	Mesh Type	Base method	Estimated complexity	Applicability to IGS
Algorithm for Counting Independent Sets on Regular and Irregular Sheets of Graphene	Irregulary Graphene Sheets	Recurrence rules+ Hamiltonian paths	$O(2^{t+1} \cdot \|E\|)$	High (designed for IGS)
Transfer matrix	Regular or semi-regular (mesh-like).	Transition matrix + vector multiplication	$O((1.618)^{(\|E\|+2)\cdot(3\|V\|)})$	Limited (requires columna pattern)
Classical methods in benzenoid systems	Benzenoid (regular, fused hexagonal type)	Topological índices, combinatorics, M-S vector	Depends on the index used (usually belongs to #P)	Low (designed for pure benzenoid structures)

products of valid configurations. Our method is based on a combination of decomposition by connected components, Hamiltonian path detection, and hierarchical recursion, allowing it to address the complete topology of an IGS without losing accuracy.

Regarding computational complexity, the proposed approach exhibits a computational behavior of order $O(2^{t+1} \cdot |E|)$, where t is the maximum number of hexagons in a row of IGS, and $|E|$ is the number of edges. This formulation enables natural parallelization through active threads operating on front edges, optimizing computation for parallel processing architectures. In $O((1.618)^{(|E|+2)\cdot(3|V|)})$, where $|E|$ is the number of edges and V would be the number of vertices. However, its applicability becomes severely limited in sheets with topological irregularities or non-periodic distributions of hexagons. Finally, traditional methods used in benzenoid systems exploit the regularity and symmetry of benzene-derived structures.

These include the use of topological indices, combinatorial algorithms, and specialized models that allow obtaining exact or approximate results in reasonable times, but are not generalizable to IGS-type meshes. Table 3 summarizes these key differences regarding structural requirements, graph coverage, computational complexity, and applicability in real-world systems. It is emphasized that the algorithm proposed here offers an exact, deterministic, and flexible solution, proving particularly advantageous for the structural analysis of irregular graphene, where other approaches lose validity or efficiency.

The transfer-matrix method, presents significant limitations when applied to non-uniform structures. This approach depends on clear segmentation into columns with consistent valid configurations, which is difficult to ensure in irregular topologies. Its use is more suitable in regular or periodic structures, such as certain crystal lattices or graphs derived from linear polymers.

Conversely, traditional approaches used in benzenoid systems (e.g., calculating the Merrifield–Simmons vector) have been specifically designed for structures with high symmetry and predictable hexagonal fusions. Although effective and with a solid combinatorial basis, these methods cannot be directly applied to IGS without requiring complex adaptations.

The results obtained demonstrate that the proposed algorithm exhibits an exact match with the traditional method in terms of total count, while also maintaining a clear advantage in execution time. In irregular topologies, the recurrence rule shows greater adaptability without the need to reconfigure matrices, making it a more robust option for defective structures common in real graphene sheets.

Comparative analysis of different approaches for counting independent sets reveals substantial differences in applicability, efficiency, and generalization across graphene-like molecular structures. The proposed recursive algorithm, based on identifying Hamiltonian paths and recursive component decomposition, demonstrates remarkable adaptability to irregular systems like irregular graphene sheets (IGS). Its

ability to operate on graphs without structural regularity makes it a particularly useful tool in contexts where other methods fail due to the absence of repetitive patterns.

7 Conclusions

We have introduced a novel algorithm for counting independent sets on regular and irregular graphene sheets, addressing the presence of topological defects and their impact on the material's structure. The implementation of the proposed algorithm is based on Fibonacci recurrence rules and the use of memoization. It is demonstrated that the application of the Fibonacci rule and the use of the Hamiltonian path for the structured traversal of the hexagonal mesh minimizes the number of operations required for calculating the number of independent sets.

The processing of defective cycles (pentagons and heptagons) is managed through load and propagation techniques, resulting in an optimized calculation structure that avoids redundancies and optimizes memory usage. This model represents a significant advancement in the study of nanomaterials, providing a robust and efficient framework for the characterization and optimization of networks with topological defects in graphene and other similar systems.

References

1. **Calkin, N. J., Wilf, H. S. (1998).** The Number of Independent Sets in a Grid Graph. SIAM Journal on Discrete Mathematics, Vol. 11, No. 1, pp. 54–60. Doi: 10.1137/s089 548019528993x.
2. **Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2009).** Introduction to algorithms. MIT Press.
3. **De Ita, G., Bello, P., Contreras, M. (2023).** A Method for Computing the Merrifield–Simmons Index on Benzenoid Systems. MATCH Commun. Math. Comput. Chem., Vol. 89, No. 1, pp. 245–Doi: 10.46793/match.89-1.245.
4. **De Ita, G., Bello, P., Marcial, R. (2024).** Counting rules for computing the number of independent sets of a grid graph. Mathematics, Vol. 12, No. 6, pp. 922. Doi: 10.3390/math12060922.
5. **Doslic, T., Zubac, I. (2015).** Saturation number of benzenoid graphs. Match communications in Mathematical and in Computer Chemistry, Vol. 73, No. 3, pp. 491–500.
6. **El-Basil, S. (1988).** Theory and computational applications of Fibonacci graphs. Journal of Mathematical Chemistry, Vol. 2, No. 1, pp. 1–29.
7. **Fei, Y., Liu, J. (2022).** Synthesis of Defective Nanographenes Containing Joined Pentagons and Heptagons. Advanced Science, Vol. 9, No. 19. Doi: 10.1002/advs.202201000.
8. **Greenhill, C. S. (2000).** The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. Computational Complexity, Vol. 9, No. 1, pp. 52–72.
9. **Goft, A., Abulafia, Y., Orion, N., Schochet, C. L., Akkermans, E. (2023).** Defects in graphene: A topological description. Physical Review. Physical Review, Vol. 108, No. 5. Doi: 10.1103/physrevb. 108.054101.
10. **Gutman, I. (2018).** Topological indices and irregular measures. Journal of Bulletin, Vol. 8, pp. 469–475.
11. **Kim, J., Han, J.-W., Yamada, Y. (2021).** Heptágonos en el plano basal de nanoláminas de grafeno analizados mediante espectroscopia fotoelectrónica de rayos X simulada. ACS Omega, Vol. 6, No. 3, pp. 2389–2395. Doi: 10.1021/ACS OMEGA.0C05717.
12. **Kwun, Y., Ali, A., Nazeer, W., Ahmad, C., Kang, S. (2018).** M-polynomials and degree-based topological indices of triangular, hourglass, and jagged-rectangle benzenoid systems. Journal of Chemistry, Article 8213950.
13. **Merrifield, R. E., Simmons, H. E. (1989).** Topological methods in chemistry. Wiley.
14. **Oz, M., Cangul, I. (2021).** Computing the Hosoya and the Merrifield-Simmons indices of two special benzenoid systems. Iranian Journal of Mathematical Chemistry, Vol. 12,

No. 3, pp. 161–174. Doi: 10.22052/ijmc.2021.243008.1580.

15. **Oz, M. S., Cangul, I. N. (2022).** Computing the Merrifield-Simmons indices of benzenoid chains and double benzenoid chains. Journal of Applied Mathematics and Computing, Vol. 68, pp. 3263–3293. Doi: 10.1007/s12190-021-01659-x.
16. **Xuan, Y., Zhao, L., Li, D., Pang, S., An, Y. (2023).** Recent advances in the applications of graphene materials for the oil and gas industry. RSC Advances, Vol. 13, pp. 23169–23180. Doi: 10.1039/ d3ra02781c.

Article received on 06/06/2025; accepted on 03/10/2025.
**Corresponding author is Marlene Mijangos Romero.*