ISSN 2007-9737

From Exponential to Linear Time: Counting Independent Sets on Polygonal Arrays

Herlinda González Vázquez[1,*], Cristina López Ramírez[1], Pedro Bello López[2], 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

231H18004@alumno.ujat.mx, cristina.lopez@ujat.mx, pedro.bello@correo.buap.mx, deitaluna63@gmail.com

Abstract. This paper compares two approaches with different time complexities for counting independent sets on arrays of molecular graphs. A known method with exponential complexity of order O(2^n), where n is the number of polygons in the array. Meanwhile, we propose a novel optimized approach based on the construction of a Hamiltonian path *Hc* on the array of polygons. This approach combines the Fibonacci and backward path rules to achieve a time complexity of order linear O($4{\cdot}n$), where n is the number of vertices in the array. This novel method not only drastically reduces the computational complexity but also demonstrates its practical applicability in the modeling and analysis of molecular graphs, such as benzenoid compounds, facilitating accurate estimates of the molecular properties and enabling the design of innovative materials.

Keywords. Polygonal arrays, molecular graphs, Merrifield-Simmon index, back edges, backward path.

1 Introduction

The Merrifield-Simmons index $i(G)$ of a graph G was introduced in 1981, and its application in mathematical chemistry was consolidated in 1989 (Merrifield & Simmons, 1981, 1989). This topological index is used to model chemical structures through graph theory, where molecules are represented as graphs, assigning vertices to atoms and edges to chemical bonds. This approach facilitates the analysis of molecular properties using topological indices, such as $i(G)$, which function as numerical descriptors by correlating chemical structures with physical and chemical properties. A publication in the American Journal of Mathematics, (Sylvester, 1878) introduced the term chemicograph to refer to the graphical notation used in chemistry.

In graph theory, $i(G)$ represents the number of subsets within the set of vertices where there is no adjacency between two vertices, i.e., the number of independent vertices that G has. $i(G)$ is also known as the Fibonacci number of a graph (Sylvester, 1878; Yurttas et al., 2020).

The computation of independent sets is important because they relate to optimization problems, such as resource planning and allocation. In addition, through $i(G)$ it is possible to determine the temperature at which the bonds of certain chemical compounds (G) break, knowing their boiling points.

This process is essential for identifying the physicochemical properties of compounds with hexagonal structures, such as graphene plates, and aromatic compounds, such as benzenes.

Independent sets are the subjects of study mainly in the field of computer science because graphs represent a powerful tool for modeling real-life problems. An example of this is the efficient allocation of classrooms and resources in educational institutions, a challenge that requires

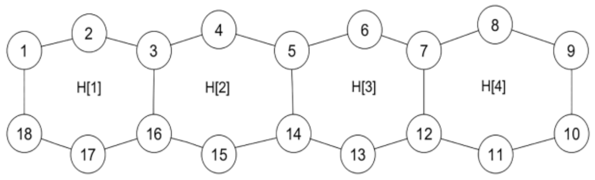

Fig. 1. One-dimensional array *H[1...n]* in graph *G*

coordinating schedules, numbers of students, teachers, and specific requirements of each class, ensuring that quality criteria are met in the distribution (López et al., 2003).

The need for efficient organization and allocation of resources is also applied in various fields, such as work team management, computer network connectivity, and activity planning.

In the field of Physics, (Bonilla, 2019) addressed the measurement of gas entropy, which could imply that, in a system such as benzenes, entropic contributions could influence molecular interactions and stability, where particles cannot share the same edge; while in chemistry, independent sets are used to model certain structural aspects of benzenoid systems (Gutman, 1982).

Therefore, graph theory and mathematical combinatorics provide solutions applicable in fields ranging from education to science, highlighting their relevance in a variety of contexts.

The contribution of this research is an efficient algorithm that performs independent set counting in polygonal arrays, combining Hamiltonian traversals and backward trajectories, reducing the order of complexity from $O(2^n)$ to $O(4\cdot n)$.

This advance contributes to molecular modeling, opening up new applications in computational chemistry, nanotechnology, and materials design.

The general structure of the paper is described below: Section 1 provides an introductory overview of the topic. Section 2 establishes the preliminary concepts and notation used throughout the paper. Section 3 describes a method based on backtracking edges for computing independent sets. Section 4 introduces a novel method that incorporates backtracking paths for such counting. Finally, Section 5 presents the conclusions and future work.

2 Preliminary

Let $G = (V, E)$ be a simple undirected graph, where V is the set of vertices and E is the set of edges. The relationship between the two vertices u and v is denoted uv, and it can also be represented with the notation $\{u, v\}$ to refer to the edge connecting both vertices. Two vertices are considered adjacent if an edge connects them. We define $N(x)=\{y \in V: \{x, y\} \in E\}$ as the neighborhood of $x \in V$. $N[x]=N(x) \cup \{x\}$ represents the closed neighborhood of x. We use $|A|$ to denote the cardinality of the set A. Similarly, the degree of vertex x is denoted as $\delta(x)=|N(x)|$, while the degree of graph G is $\Delta(G)=max\{\delta(x): x \in V\}$. The set $S \subset V(G)$ of the vertices of G is an independent set in G, if for any pair of vertices $u,v \in S$, it is satisfied that $\{u, v\}$ is not in E. Denoted by $I(G)= \{S/S$ is an independent set in $G\}$ to the set of all independent set in G. Meanwhile, $i(G)=|I(G)|$ is the number of independent sets in G. Let $v \in V(G)$, denoted by $I_v(G) = \{S \in I(G): v \in S\}$, $I_{-v}(G) = \{S \in I(G): v$ is not in $S\}$.

Several methods have been developed for the computation of the number of independent sets in planar lattice structures that simplify the counting process using paths organized by rows and columns (De Ita et al., 2023). In this paper, our proposal is a method to compute the Merrifield-Simmons (MS) index; on a one-dimensional array of polygons *H[1...n]* consisting of *n* polygons, as shown in Figure 1. We differentiate two independent set computation processes: counting independent sets with Hamiltonian path *Hc* in *H[1...n]* with back edges and counting with backward paths.

In (De Ita et al., 2024), two counting rules have been proposed: the Fibonacci rule (1), used for counting on paths, and the subtraction rule (2), applied to counting on the back edge of a cycle. The process of counting independent sets for a one-dimensional array of polygons starts with a main thread (*Lp*), where the initial pair of values (called the charge) is: (α_1, β_1)= (1,1). Similarly, when they are polygons, a secondary thread (*Ls*) starting with a charge (α_1, β_1)= (0, 1) is used. The series of values (α_i, β_i), $i=1,...,m$ is identified as a counting thread, then *Lp* and *Ls* are two counting threads used to store the charges of the vertices visited during the Hamiltonian trajectory on the

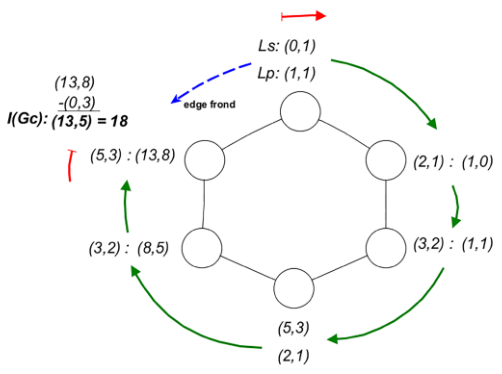

Fig. 2. Independent set counting

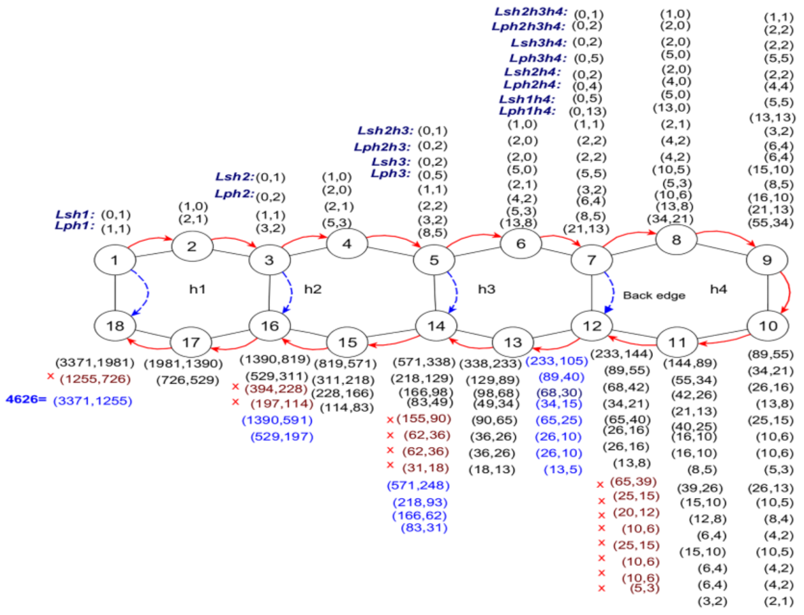

Fig. 3. Counting independent sets with an exponential time-complexity

array. The idea is traversing by the array of polygons through a Hamiltonian path. The traversing recognizes two types of edges: tree edges and back edges. The Fibonacci rule (1) is applied when a tree edge is visited. Meanwhile, the subtraction rule (2) is applied for back edges:

$$(\alpha vi+1, \beta\, vi+1) : \alpha\, vi+1 = \alpha vi + \beta vi \;; \quad \beta\, vi+1 = \alpha vi\,, \tag{1}$$

$$\begin{aligned}(\alpha w, \beta w) i &= (\alpha w\,, \beta w\,) - (0, \beta v w\,) \\ &= (\alpha w\,, \beta w\, - \beta v, w).\end{aligned} \tag{2}$$

Figure 2 shows how the charge associated to each vertex is modified according to a tree edge or a back edge is visited. And the start of the count is indicated by an arrow (⟶) (De Ita et al., 2020). At the end of the Hamiltonian path, both threads (*Lp* and *Ls*) have charges (13,8) and (5,3), respectively. The last pair of the secondary thread *(α_m, β_m) = (0, β_m)* corresponds to the back edge *{v_n,*

v_0} ∈ E(G), which closes the cycle. This value is subtracted from the last pair of the main thread *Lp*. Thus, *((α_m, β_m) - (0, β_m)) = ((α_6, β_6) - (0, β_6)) = (13, 8) - (0, 3) = (13, 5)*. Therefore, *i*(*G*)= *13 + 5 = 18*, which gives a total of 18 independent sets.

3 Method to Compute Independent Sets with Back Edges

To illustrate the method to compute independent sets with back edges, we present an example in Figure 3, considering an one-dimensional array of polygons.

Starts with a Hamiltonian path *Hc* (illustrated in red color), traversing through a one-dimensional array of polygons, and each vertex is visited only once. The array of polygons is formed by 4 hexagons and to process a row, the Fibonacci (1) and subtraction (2) rules are sufficient, since only tree edges and only one back edge for each polygon are identified. The computation of *i(G)* starts at vertex 1 and is processed up to vertex 18.

In this process, a main thread *Lp* starts with the initial value pair *(α_1, β_1)* = (1,1), while the secondary thread *Ls* signals the start of each of the polygons with the sequence *(α_1, β_1)* = (0, 1). Both threads, *Lp* and *Ls*, advance simultaneously in the computation following the Fibonacci recurrence rule of equation (1). Upon detecting the start of a new polygon, new threads *(Lph1...n)* and *(Lsh1...n)* are generated, which doubles the number of active threads in parallel for each bifurcation encountered. When a back edge is identified (illustrated in blue color), the corresponding polygon *(Ls)* is closed by applying the subtraction rule of equation (2), which allows updating the charge of the secondary threads and reducing the number of active threads. To close each polygon of the array, the last pair *(α_m, β_m)* of the secondary thread *(Lsh1...n)* is defined as *(α_m, β_m) = (0, β_m)*, where the edge *{v_m, v_0} ∈ E(G)* represents the back edge. This pair is subtracted from the last pair in the primary thread of *(Lsh1...n)*. This process is repeated iteratively, closing each polygon of the one-dimensional array such that the back edges of *(Ls)* are encountered until the Hamiltonian path is complete.

The number of independent sets in a one-dimensional array of polygons can be obtained using the Hamiltonian trajectory *Hc* as a reference. However, the complexity of this process grows exponentially because it depends on the maximum number of back edges due to the cycles that remain open during the trajectory. Although computing *i*(*G*) still has exponential time complexity, this approach, unlike classical methods such as the transfer matrix (Euler, 2005), does not exhibit such explosive combinatorial growth. Determining the number of independent sets in a molecular graph, as in the case of aromatic compounds, such as benzenes, helps to estimate their boiling temperatures. This information allows us to not only model more efficient compounds but also design new molecules with different chemical properties. However, the time complexity is order *$O(2^n)$*, where *n* is the number of polygons in the array. This results in an exponential procedure for computing *i*(*G*) because of the *n* polygons that remain open along the path *Hc* on in *G*.

On the other hand, (De Ita et al., 2024, pp. 12–13) present the time complexity analysis of the transfer matrix method for this kind of arrays. For example in a grid of *m* columns and *n* rows, only the construction of T_m requires an order $O((F_{m+2})^2 \cdot (m+1))$ operations, where F_{m+2} represents the Fibonacci number at position *m+2*, and each dot product involves vectors of length *m*+1. Consequently, this method exhibits exponential complexity with respect to the grid dimensions *m* and *n*.

Machine learning techniques for counting independent sets in graphs have gained ground due to the inherent complexity of the problem. Pontoizeau et al., (2021) presented a proposal to solve the MAX INDEPENDENT SET problem by combining Graph Neural Networks (GNN) with the Expert Iteration technique, which promotes future advances in artificial intelligence applied to complex problems. Sungsoo et al., (2020) present another approach to solve the Maximum Independent Set (MIS) problem using a new deep reinforcement learning (DRL) framework called Learn what to delay (LwD). Unlike traditional DRL methods, which make decisions in a fixed number of stages, LwD allows dynamically adapting the time at which each decision is made, which improves efficiency in large-scale graphs. Traditional methods do not allow for exact

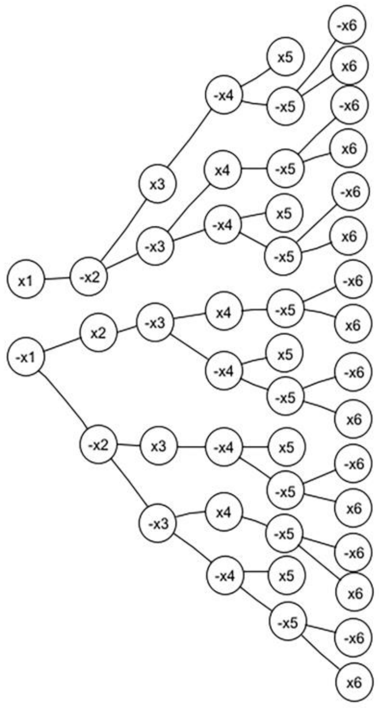

Fig. 4. Binary tree to construct the backward path rule

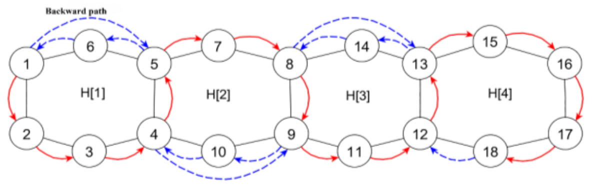

Fig. 5. A Hamiltonian path *Hc* on the array: *H*[1...*n*] with backward paths

computations, leading researchers to explore approximation algorithms and innovative computational strategies, thus opening the door to new research at the intersection of machine learning and combinatorial optimization.

4 A Novel Method to Compute Independent Sets Considering a Backward Path

We introduce a novel method for the computation of independent sets, which is simplified by focusing on three main aspects: the back edges, the backward path, and the process required to return to the starting point of the path. To construct the backward path rule, is based on the inclusion-exclusion method derived from the following binary tree method (Figure 4).

Let us consider the binary tree (see Figure 4), states can be defined based on the inclusion-exclusion of a vertex in any independent set, thus optimizing the calculation of the total number of independent sets (Aleid et al., 2018; Law, 2010). The process starts with the construction of the binary tree at $-x_1$ and x_1, which represent the initial pair (α_1, β_1) = (1, 1). This pair establishes the starting point for counting the independent sets in *Lp* of the graph fragment shown in Figure 6. Such a start indicates that the counting starts with an empty set and a single vertex. This binary tree facilitates the construction of pairwise combinations in which an independent set is composed of non-adjacent vertices, and therefore, the paths on the tree avoid two consecutive positive vertices. The negative symbol (-) indicates that a vertex cannot be in a set. For example, upon reaching vertex x_5, counting on the main line *Lp* with (α_5, β_5) = (8, 5) signals that 8 pairs of vertices cannot be part of the set, denoted as $-x_5$, while 5 pairs of vertices can be included, denoted as x_5.

The proposed method guarantees an efficient process for computing independent sets in *G*, resulting in the creation of a backward path (3) rule that can be applied to a path with *2, 3, 4, ...,* or *n* edges. In the case of hexagonal arrays, the backward path includes only two back edges. In the backward path, calculations are not saved during the traversal; instead, they are processed instantly to continue with the processing of the next polygon and traverse all polygons until the entire graph *G* is finished. Figure 5 shows how the identified backward paths are processed as the route proceeds. To start counting the Merrifield-Simmons index, we consider the two previous rules (1) the Fibonacci rule and (2) the Back edge rule, and the new rule: the Backward path rule. When we consider two edges forming the backward path and if at the beginning of the backward the charge (αp, βp) is in *Lp*, and the charge (αs, βs) is in *Ls*, then the rule (3) derived from the binary tree, is:

$$(\alpha p, \beta p) = (2\alpha p - \alpha s, \quad \beta p). \tag{3}$$

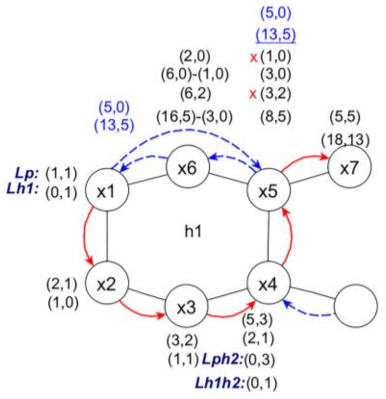

Fig. 6. Counting independent sets with a backward path

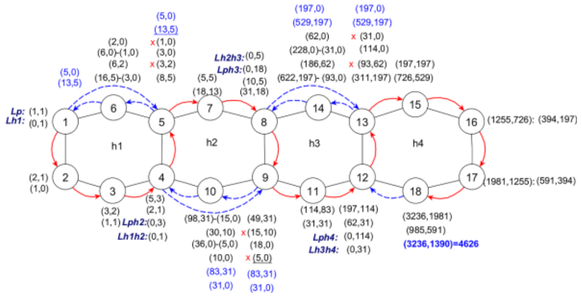

Fig. 7. Efficient computation of the number of independent sets in a hexagonal array

The counting rule (3) updates the charge for any main thread *Lp* and it closes any subordinated thread (indicated with a red X at the beginning of the charge of *Ls,* in Fig. 6). In Fig. 5, when an array of hexagons is processed, the Hamiltonian path *Hc* (illustrated in red color) starts in vertex 1 until achieve a backward path (illustrated in blue color) formed by two back edges, and then the rule (3) is applied, while the independent sets are computed simultaneously.

However, to complete the computation of the polygonal array *G* and close, a subtraction operation corresponding to the backward path must be added. This subtraction allows the calculation to be properly adjusted and ensures accuracy in the final accounting of the independent sets.

The Hamiltonian path (red lines) follows a linear trajectory covering a line of polygons, as illustrated in Figure 5, eliminating the need to handle the exponential complexity observed in previous approach. In the worst-case scenario, this method requires only the construction of 4 processing lines, thereby avoiding the exponential: 8, 16, 32, etc., computing threads that is a characteristic of other less efficient methods. This computational saving is achieved by restricting the operations to what is strictly necessary, which makes this approach a practical and efficient alternative for counting independent sets on polygonal arrays.

The main thread (Lp), starts *Lp:* $((\alpha_1, \beta_1) = (1, 1) \rightarrow (2\alpha_2, \beta_2) = (2, 1) \rightarrow (3\alpha_3, 2\beta_3) = (3, 2) \rightarrow (5\alpha_4, 3\beta_4) = (5, 3) \rightarrow (8\alpha_5, 5\beta_5) = (8, 5) \rightarrow \ldots \rightarrow (F\alpha{+}1\alpha,\ F\alpha\ \beta) = (\alpha m,\ \beta m))$. Meanwhile, the secondary thread (Ls) computes the value for $|\{S \in i(G') : v1 \in S \wedge vm \in S\}|$ in *G*. *Ls:* $(\alpha s, \beta s) = (0, 1) \rightarrow (1\alpha s{+}\beta s, \alpha s) = (1, 0) \rightarrow (2\alpha s{+}\beta s, \alpha s{+}\beta s) = (1, 1) \rightarrow (3\alpha s{+}2\beta s,\ 2\alpha s{+}\beta s) = (2, 1) \rightarrow (5\alpha s{+}3\beta s,\ 3\alpha s{+}2\beta s) = (3, 2) \rightarrow \ldots \rightarrow (F\alpha{+}1\alpha - F\alpha{-}1\alpha s, F\alpha\beta - F\alpha{-}2\beta) = (\alpha m, 0))$, to close the open cycles in each polygon. Finally, the general process series for counting independent sets considering a backward path of length *k is* $(\alpha, \beta{-}\beta s) \rightarrow (2\alpha{-}\alpha s, \beta) \rightarrow (3\alpha{-}\alpha s, \beta - \beta s) \rightarrow (5\alpha{-}2\alpha s, 3\beta - \beta s) \rightarrow (8\alpha{-}3\alpha s, 5\beta{-}2\beta s) \ldots$ where $k = 1, 2, 3, \ldots, n$ back edges.

The Backward path rule (3) allows a drastic reduction in the number of required computational lines, considerably reducing the complexity order compared to traditional methods, as with the transfer matrix method (Hosoya, 1973), and overcomes the exponential complexity associated with a row of polygons with backward paths with only one back edge (De Ita et al., 2024). As a result, the method operates with a linear order complexity of $O(4\cdot n)$, where *n* is the number of vertices to visit, multiplied by 4 which are the number of computing threads to use, and which represents a more manageable and efficient solution for computing *i*(*G*).

In Figure 6, we use a section of the graph presented in Figure 7 to illustrate the counting of independent sets on only one hexagon and by applying the backward path rule (3). Table 1 summarizes the counting of independent sets on tree edges, and considering the backward path.

We start the computation of the number of independent sets of graph *G* from vertex x1. This starting point marks the start of the path, allowing us to apply the established rules to identify and count the independent sets as the vertices and edges of the graph are explored. The counting

Algorithm 1. Backward paths

```
Input: matrix, list
Output: back_path

Function backward_path (matrix, list):
    initialize back_path ← empty_list
    initialize list_edge ← empty_list
    for each row i in matrix do
        for each column j in matrix[i] do
            if matrix[i][j] == 1 then
                start ← list.search(i)
                end ← list.search(j)
                if start ≠ none and end ≠ none then
                    if(start.data,end.data)not in list_edge and (end.data, start.data)
                      not in list_edge
                        add(end.data, start.data)to list_edge
                        previous.aux ← end.next
                        end.aux ← start
                        start.aux ← previous
                                                                          add (previous.data,
                                                    end.data, start.data)to back_path
                    end_if
                end_if
            end_if
        end_for
    end_for
    return back_path
End_function
```

starts with a main thread *Lp* starting with the pair of values with *(α_1, β_1)* = (1, 1), and the secondary thread *Lh1* marking the start of polygon *h1* with the sequence *(α_1, β_1)* = (0, 1). Both threads *Lp* and *Lh1* advance in parallel by applying the Fibonacci recurrence rule (1) on the tree edges.

This process continues until it finds the beginning of a backward path in vertex x4. It duplicates the threads of execution because it indicates the start of a new polygon. The walk continues until it finds the start of a backward path to be visited, formed by x5-x6-x1 and which has to return to x5, closing polygon *h1.* Then, the backward path rule (3) has to be applied after visit x5, and the Hamiltonian walk continues to the next polygon *h*2.

Our method uses four computing threads for a hexagonal array, and ensures that each vertex is visited only once. This approach simplifies complex calculations and adapts well to various applications in graph theory and related fields. The usefulness of the method is highlighted by its ability to determine the number of independent sets in compounds with specific structures, such as benzenoid systems. In the case of aromatic compounds, such as benzene, knowing the number of independent sets provides key information about how the molecular bonds behave under different physical and chemical conditions. For example, this analysis makes it possible to establish the relationship between the independent sets and the critical temperatures at which the chemical bonds are detached, which in turn is linked to the decomposition and boiling temperatures of these compounds. This is especially relevant in industrial and scientific contexts where the aim is to optimize processes related to chemical materials, such as the creation of new compounds or the design of more efficient materials.

The number of polygons in an array are essential for solving graph embedding problems and analyzing polygon meshes (González et al., 2024). In our study, a one-dimensional array of hexagons has been considered. Therefore, to perform a path in linear order, it is essential identify the faces that constitute the polygonal array. In tables 2 and 3 present the numerical calculations for counting independent sets, clearly demonstrating the difference in the number of

Algorithm 2. Linear order path for counting independent sets

```
Input: list_edges
Output: path_ham, i(G)
Function route(list_edges):
    if head_of_list is null then
        return empty_list
    end_if
    global path_ham
    start_vertice ← head_of_list
    end_vertices ← null
    current ← ini_vertice
    path_ham ← empty_list
    i ← 0
    i(G) ← 0
    while current is not null and i < length(back_path) do:
        add current.data to path_ham
        if current.data ≠ back_path [i][0] and end_vertice is null then
            #rule 1.    ▷ This is a comment
            i(G) ← i(G) + call Function Fibonacci()
            previous ← current
            current ← current.next
        else if current.data == back_path [i][0] and end_vertice is null then
            #rule 1.    ▷ This is a comment
            i(G) ← i(G) + call Function Fibonacci()
            fin_vertice ← current.next
            current ← current.next
     else if current.data == back_path [i][2] and end_vertice is not null then
         #rule 2.    ▷ This is a comment
         i(G) ← i(G) + call Function Back_Edge()
            current ← current.aux
     else if current.data == back_path [i][1] and end_vertice is not null then
         #rule 3.  ▷ This is a comment
         i(G) ← i(G) + call Function Backward_Path()
            current ← current.aux
        else:
            #Start backward path    ▷ This is a comment
            current ← current.aux
            end_vertice ← null
            i ← i + 1
        end_if
    end_while
    i(G) ← i(G) + call Function Back_Edge()
    #rule 2 to close of G with {previous.dato, previous.aux.dato} ▷ This is comment
    add previous.aux.dato to path_ham
    print path_ham
    return path_ham, i(G)
End_function
```

operations required for "Counting Independent Sets with Exponential Time Complexity" and the approach proposed in this work, "Efficient Computation of the Number of Independent Sets."

This comparison demonstrates the significant improvement in the computational efficiency.

The algorithm for counting independent sets in a polygonal array, using backward paths based on the previously described calculation method is detailed below. The general algorithm follows the following steps:

– Construction of the adjacency list: the adjacency list is generated from the input data to represent the connections between the vertices of the graph.

– Creation of the graph and adjacency matrix: The graph *G* and its adjacency matrix *M* are constructed by assigning labels to the vertices

Table 1. Counting independent sets

x1	x2	x3	x4	x5	x5→x6→x1→x5	
Lp: $(\alpha_1,\ \beta_1) \rightarrow$	$(2\alpha_2,\ \beta_2)$	$(3\alpha_3,\ 2\beta_3) \rightarrow$	$(5\alpha_4,\ 2\beta_4) \rightarrow$	$(8\alpha_5,\ 5\beta_5) \rightarrow$		
Ls: $(\alpha_s,\ \beta_s) \rightarrow$	$(\alpha_s,\ \beta_s)$	$(\alpha_s,\ \beta_s) \rightarrow$	$(2\alpha_s,\ \beta_s) \rightarrow$	$(3\alpha_s,\ 2\beta_s) \rightarrow$		
Lp: $(1,1) \rightarrow$	$(2,1)$	$(3,2) \rightarrow$	$(5,3) \rightarrow$	$(8,5) \rightarrow$		
Lph1: $(0,1) \rightarrow$	$(1,0)$	$(1,1) \rightarrow$	$(2,1) \rightarrow$	$(3,2) \rightarrow$		
		Lph2	$(0,3) \rightarrow$	$(3,0) \rightarrow$		
		Lh1h2	$(0,1) \rightarrow$	$(1,0) \rightarrow$		
				$(\alpha p, \beta p - \beta s) \rightarrow$	$(2\alpha p - \alpha p, \quad \beta p)$	
				$(8,5) \rightarrow$	$(16-3,5) =$	$(13,5) \rightarrow$
				$(3,2) \rightarrow$	$(6,2)\ x \rightarrow$	
				$(3,0) \rightarrow$	$(6-1,0) =$	$(5,0)$
				$(1,0) \rightarrow$	$(2,0)\ x \rightarrow$	

in the rows and columns and marking the adjacent connections with 1.

– Application of *DFS (Depth-First Search)*: A depth-first search is performed from an initial vertex. The visited vertices are recorded in a linked list and marked as 0 in the adjacent matrix.

– Identification of backward paths: During the traversal of the matrix, the pairs of adjacent vertices marked with 1 identify the back edges. These edges are linked in the simple linked list of visited vertices, and a vertex preceding a pair of backtracking edges is included as an additional element in the backward path. This additional element is added as part of the backward path and is also linked in the linked list for subsequent traversals.

– Cycle Detection and Faces Counting: The linked list was analyzed to identify closed cycles. The vertices that are part of each cycle are added to the list of faces and the total number of faces in graph *G*.

– Path of the graph for counting the independent sets: A path is constructed following the linked list constructed in step 3° using pointers to traverse the vertices of the graph.

Algorithm 1, "Backward paths"; identifies and constructs backward paths in a graph from an adjacency matrix and a linked list of visited vertices. This algorithm is useful for graph processing and finding backward paths. The process is as follows:

– Initializes lists to store backward paths and edges of the graph.

– The adjacency matrix is searched for pairs of connected vertices (i and j) where an edge exists (matrix[i][j] == 1).

– The vertices are verified as valid, and the edge has not been previously registered.

– If the conditions are satisfied, the linked list, auxiliary (aux) links, and the backward path are saved.

– Finally, it returns to the list of backward paths.

Algorithm 2, "Linear Order Path for Counting Independent Sets", is designed to construct and return a path (stored in *path_ham*) in a graph using a list of edges and backward paths previously defined in back*_path*.

The path function checks whether the list is empty. If so, the list returns to an empty one. Then, it initializes the following key variables: *ini*_vertice: points to the start of the linked list; *fin*_vertice: marks the end of a subpath (initially NULL); *current*: functions as a pointer to traverse the list; *path_ham*: is the list intended to store the constructed path. The traversal process starts

Table 2. Counting independent sets with an exponential time-complexity

		1		2		3		4		5		6		7		8		9		10		11			12					13			14				15				16					17				18		
Lph$_1$	1	1	2	1	3	2	5	3	8	5	13	8	21	13	34	21	55	34	89	55	144	89	233	144	0	39	233	105	338	233	571	338	0	90	571	248	819	571	1390	819	0	228	1390	591	###	1390	3371	1981	0	726	3371	1255
Lsh$_1$	0	1	1	0	1	1	2	1	3	2	5	3	8	5	13	8	21	13	34	21	55	34	89	55	0	15	89	40	129	89	218	129	0	36	218	93	311	218	529	311	0	144	529	197	726	529	1255	726	X		**CI =**	**4626**
Lph$_2$					0	2	2	0	2	2	4	2	6	4	10	6	16	10	26	16	42	26	68	42	0	12	68	30	98	68	166	98	0	36	166	62	228	166	394	228	X											
Lsh$_2$					0	1	1	0	1	1	2	1	3	2	5	3	8	5	13	8	21	13	34	21	0	6	34	15	49	34	83	49	0	18	83	31	114	83	197	114	X											
Lph$_3$									0	5	5	0	5	5	10	5	15	10	25	15	40	25	65	40	0	15	65	25	90	65	155	90	X																			
Lsh$_3$									0	2	2	0	2	2	4	2	6	4	10	6	16	10	26	16	0	6	26	10	36	26	62	36	X																			
Lph$_2$h$_3$									0	2	2	0	2	2	4	2	6	4	10	6	16	10	26	16	0	6	26	10	36	26	62	36	X																			
Lsh$_2$h$_3$									0	1	1	0	1	1	2	1	3	2	5	3	8	5	13	8	0	3	13	5	18	13	31	18																				
Lph$_1$h$_4$													0	13	13	0	13	13	26	13	39	26	65	39		X																										
Lsh$_1$h$_4$													0	5	5	0	5	5	10	5	15	10	25	15		X																										
Lph$_2$h$_4$													0	4	4	0	4	4	8	4	12	8	20	12		X																										
Lsh$_2$h$_4$													0	2	2	0	2	2	4	2	6	4	10	6		X																										
Lph$_3$h$_4$													0	5	5	0	5	5	10	5	15	10	25	15		X																										
Lsh$_3$h$_4$													0	2	2	0	2	2	4	2	6	4	10	6		X																										
Lph$_2$h$_3$h$_4$													0	2	2	0	2	2	4	2	6	4	10	6		X																										
Lsh$_2$h$_3$h$_4$													0	1	1	0	1	1	2	1	3	2	5	3		X																										

Table 3. Efficient computation of the number of independent sets

	1		2		3		4		5		5→6→1 →5		7		8		9		9→10→4 →9		11		12		13		13→14→8→13			15		16		17		18			
LpLh$_1$	1	1	2	1	3	2	5	3	8	5	16-3, 5	13, 5	18	13	31	18	49	31	98-15, 31		114	83	197	114	311	197	622-93, 197	529	197	726	529	1255	726	1981	1255	3236	1981	3236	1390
	0	1	1	0	1	1	2	1	3	2	6,2	X														0												**CI =**	**4626**
Lph$_2$							0	3	3	0	6-1, 0	5, 0	5	5	10	5	15	10	30, 10	X						0													
Lph$_1$h$_2$							0	1	1	0	2, 0	X														0													
Lph$_3$															0	18	18	0	36-5, 0	31, 0	31	31	62	31	93	62	186, 62	X											
Lh$_2$h$_3$															0	5	5	0	10, 0	X						0													
Lph$_4$																							0	114	114	81, 0	228-31, 0	197	**0**	197	197	394	197	591	394	985	591		
Lh$_3$h$_4$																							0	31	31	62, 0	62, 0	X											

when the variables are initialized. If vertices are to be processed and all backward paths have not been completed, the algorithm performs the following actions:

- The current vertex data (current.data) are added to the path_ham list.
- According to the conditions related to the current vertex (current.data) and the backward paths defined in back_path, the current pointer is set to advance to the next vertex in the linked list or follow the auxiliary link (aux).
- Controls the state of *fin_*vertice to detect the start or end of a subpath and, if appropriate, increments index *i* to move to the next backward path. Finally, it prints the *path_ham* list and returns the constructed path as the result. Finally, it can be observed that path_ham lists the entire graph and that it has been performed in linear order at the same time as the counting of the independent sets *i(G)* is being performed.

The presented algorithms show significant improvement in counting independent sets in one-dimensional arrays of polygonal structures. Their main advantage lies in the reduction of computational complexity, which allows them to address large instances with considerably greater efficiency than classical methods, such as the transfer matrix. However, the method has clear limitations. Its applicability is restricted to linear arrays because it was designed exclusively for one-dimensional polygonal structures.

Therefore, it has to be adapted in the case of two- or three-dimensional meshes. This motivates new lines of research, such as extending the algorithm to more general polygonal meshes.

5 Conclusions and Future Work

Independent sets transcend disciplines by simplifying and optimizing calculations in complex chemical systems, thereby enabling the modeling of chemical reactions and the prediction of molecular properties in compounds with polygonal structures.

The algorithms introduced here not only enriches the understanding of chemical systems, but also opens new opportunities in areas such as nanotechnology, drug design, and advanced materials development. In particular, counting the number of independent sets ensembles in polygonal arrays using the Hamiltonian trajectory *Hc* represents a significant advance over traditional methods, such as the transfer matrix method.

The application of this method to benzenoid compounds allows the estimation of key molecular properties, such as boiling temperature, enabling the design of new materials with innovative properties. We have designed an efficient algorithm that combines Hamiltonian path and backward path rule to compute the Merrifield-Simmons index on polygonal arrays. This approach, with a time complexity reduced to linear order $O(4 \cdot n)$, where n is the number of vertices in the array, overcomes the limitations of traditional exponential growth methods, demonstrating its importance in the chemical modeling and optimization of processes related to polygonal structures.

In future work, we propose to extend this approach to the analysis of more general polygonal meshes by applying the Branch-and-Bound methodology to identify base cases and incorporate backtracking paths, and to design an exact algorithm to count independent sets within such structures.

References

1. **Aleid, S., Cáceres, J., Puertas, M. (2018).** On independent [1, 2]-sets in trees. Discuss. Math. Graph Theory, Vol. 38, No. 3, pp. 645–660. Doi: 10.7151/DMGT.2029.
2. **Bonilla, A. (2019).** ¿Podría ser la gravedad una fuerza entrópica? Tesis de Maestría, UNED.
3. **De Ita, G., Bello, P., Contreras, M. (2023).** A Method for Computing the Merrifield-Simmons Index on Benzenoid Systems. Communications in Mathematical and in Computer Chemistry, Vol. 89, No. 1, pp. 245–270. Doi: 10.46793/match.89-1.245I.
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. **De Ita, G., Rodríguez, M., Bello, P., Contreras, M. (2020).** Basic Pattern Graphs for the Efficient Computation of Its Number of Independent Sets. Springer, Vol. 12, pp. 57–66. Doi: 10.1007/978-3-030-49076-8_6.
6. **González, H., López, C., Bello, P., De Ita, G. (2024).** Algoritmo de detección de caras en grafos hexagonales: un enfoque para el conteo de conjuntos independientes. Abstraction & Application, Vol. 47, pp. 74–86.
7. **Gutman, I. (1982).** Topological Properties of Benzenoid Systems. IX*. On the Sextet Polynomial. Zeitschrift für Naturforschung A, Vol. 37, No. 1, pp. 69–73. Doi: 10.1515/zna-1982-0115.
8. **Hosoya, H. (1973).** Topological index and Fibonacci numbers with relation to chemistry. The Fibonacci Quarterly, Vol. 11, No. 3, pp. 255–266. Doi: 10.1080/00150517. 1973.12430822.
9. **Law, H. (2010).** On the number of independent sets in a tree. The Electronic Journal of Combinatorics, Vol. 17, No. 18, pp. 1–5. Doi: 10.37236/467.
10. **López, R., Gutiérrez, M., Ortuño, M., Ramírez, J. (2003).** El problema del conjunto independiente en la selección de horarios de cursos. Revista de Matemática: Teoría y

ISSN 2007-9737

Aplicaciones, Vol. 10, No. 1-2, pp. 156–167. Doi: 10.15517/rmta.v10i1-2.231.

11. **Merrifield, R., Simmons H. (1981).** Enumeration of structure-sensitive graphical subsets: Theory. Mathematics, Vol. 78, No. 2, pp. 692–695. Doi: 10.1073/pnas.78.2.692.
12. **Merrifield, R., Simmons, H. (1989).** Topological methods in chemistry. Wiley-Interscience., Vol. 221, No. 3, A521. Doi: 10.1016/0167-2584(89)90656-7.
13. **Pontoizeau, T., Sikora, F., Yger, F., Cazenave, T. (2021).** Neural Maximum Independent Set. Springer Nature. Doi: 10.1007/978-3-030-93736-2_18.
14. **Sungsoo, A., Younggyo, S., Jinwoo, S. (2020).** Learning What to Defer for Maximum Independent Sets. International conference on machine learning, Vol. 119, pp. 134–144.
15. **Sylvester, J. (1878).** On an Application of the New Atomic Theory to the Graphical Representation of the Invariants and Covariants of Binary Quantics, with Three Appendices. American Journal of Mathematics, Vol. 1, No. 1, pp. 64–104. Doi: 10.2307/2369436.
16. **Yurttas, A., Delen, S., Demirci, M., Sinan, A., Naci, I. (2020).** Fibonacci Graphs. Symmetry, Vol. 12, No. 9, pp. 1383. Doi: 10.3390/sym12091383.

Article received on 04/06/2025; accepted on 04/10/2025.
**Corresponding author is Herlinda González Vázquez.*