MATCH
Communications in Mathematical and in Computer Chemistry

MATCH Commun. Math. Comput. Chem. **79** (2018) 55-78

ISSN 0340 - 6253

Linear–time Algorithms for Computing the Merrifield–Simmons Index on Polygonal Trees

Guillermo De Ita Luna[1], **J. Raymundo Marcial–Romero**[2], **Pedro Bello López**[1], **Meliza Contreras González**[1]

[1]*Facultad de Cs.de la Computación, Benemérita Universidad Autónoma de Puebla, Puebla, México*

[2]*Facultad de Ingeniería, Universidad Autónoma del Estado de México, Toluca, México*

(Received May 13, 2017)

Abstract

We present linear-time algorithms for Computing the Merrifield-Simmons Index (counting the number of independent sets) on polygonal tree graphs, that are graphical representations of molecular graphs. Our methods combine a treewidth decomposition of the input graph, that is a central graph technique, together with the application of macros, that is a common tool used in Artificial Intelligence for representing cumulative series of basic operations.

A *macro* is associated to each basic graphic pattern in our decomposition, allowing us to represent and perform a serie of repetitive operations while the same pattern graph is found. This results in a linear-time algorithm for counting the number of independent sets on repetitive graph topologies, such as polygonal arrays. Due to the existence of an efficient treewidth decomposition on polygonal trees, it is also possible to count independent sets efficiently on this class of graphs.

1 Introduction

In hard counting problems, the computation of the number of independent sets of a graph G, denoted as $i(G)$, has been key for determining the frontier between efficient counting (polynomial-time solvable) and intractable counting problems.

The recognition of structural patterns appearing on graphs has been helpful to design efficient algorithms for computing $i(G)$. For example, the linear-time Okamoto's algo-

rithm [12] computes $i(G)$ when G is a chordal graph, and where the decomposition of G in its clique-tree gives the possibility of applying dynamic programming in an efficient way. Another case, is the Zhao's algorithm for computing $i(G)$ on regular graphs [20].

Decompositions of graphs such as clique separators, treewidth decomposition, and clique decomposition are often used to design efficient graph algorithms. There are even wonderful general results stating that a variety of NP-complete graph problems can be solved in polynomial time for graphs of bounded treewidth and bounded clique-width [7]. In order to obtain efficient algorithms based on this approach, the input graphs have to be restricted to a graph class, which has a bounded treewidth or a bounded clique-width decomposition.

Polygonal array graphs have been widely investigated, and they represent a relevant area of interest in mathematical chemistry, since they are molecular graphs used to represent the structural formula of chemical compound. In addition, it is also important to recognize substructures of those compounds and learn messages from the graphic model through the clear elucidation of their structures and properties [16].

The Merrifield-Simmons index of a molecular graph G, that is the number of independent sets of G, is a typical example of an invariant used in mathematical chemistry for quantifying relevant details of molecular structures. Merrifield and Simmons showed the correlation between $i(G)$ and boiling points on polygonal chain graphs representing chemical molecules [4, 16].

In particular, hexagonal chains are the graph representations of an important subclass of benzenoid molecules, unbranched catacondensed benzenoid molecules, which play a distinguished role in the theoretical chemistry of benzenoid hydrocarbons [15]. The propensity of carbon atoms to form compounds, made of hexagonal arrays fused along the edges, motivated the study of chemical properties of hydrocarbons via hexagonal chains. Those graphs have been widely investigated and represent a relevant area of interest in mathematical chemistry, since they are used for quantifying relevant details of the molecular structure of the benzenoid hydrocarbons [6, 15, 16].

The extremal chain graphs with respect to some useful topological indices in chemical applications, such as the Merrified-Simmons index, have been extensively studied. In 1993, Gutman [9] discussed the extremal hexagonal chains according to three topological invariants: Hosoya index, largest eigenvalue, and Merrified-Simmons index. His work

greatly motivated the study of extremal polygonal chains.

On his seminal paper, Gutman showed extremal linear chains for Merrifield-Simmons index for the particular case of hexagonal chains. He conjectured that the hexagonal chain with the smallest Merrifield-Simmons index is unique and it corresponds to the zig-zag polyphenegraph.

L.Zhang [17] showed Gutman's conjecture. They showed that the minimum value for the Merrifield-Simmons index is achieved by the zig-zag polyphenegraph. Later on, Cao et al. [2] showed extremal polygonal chains for k-matchings (Hosoya index), considering the topology of polygonal arrays that provide maximum as well as minimum values for the Hosoya index. Their demonstrations are based on the use of the Z-polynomial (Z-counting polynomial). Afterwards, Zhang, Wang and Li [18] determined the extremal hexagonal chains concerning to the total ϕ-electron energy, which are similar to the extremal chains in [19].

In recent years, several works have been done on the extremal problem for the values of those two indices, i.e., on determining the graphs within a prescribed class that minimize or maximize the value of the Hosoya and Merrifield-Simmons indices [2,4,15,16,19]. Some of those works deal with the characterization of the extremal graphs with respect to these two indices in several given graph classes. Usually, trees, unicyclic graphs, and certain structures involving pentagonal and hexagonal cycles have been analyzed [2, 4, 5, 15, 21]. In [21], a survey about extremal graphs for Hosoya and Merrifield-Simmons indices for different graph topologies is considered.

The previous works are connected to hexagonal chains, or to the Hosoya index, and the obtained results show that typically the graphs of minimal Hosoya index coincide with those of maximal Merrifield-Simmons index and vice versa. However, the correlations between these two indices are not fully understood yet. For example, Deng [5] showed that for the graphs with n vertices and $n + 1$ edges, denoted as $(n, n + 1)$-graphs, the smallest Merrifield-Simmons index do not coincide with those of the largest Hosoya index.

On the other hand, the recognition of repetitive structures 'patterns' in graphs is essential in the design of efficient algorithms for processing combinatorial objects on them. Furthermore, some tools developed in AI (Artificial Intelligence) have been useful to process combinatorial objects in graphs [1,14]. In our case, we will apply *macros* represented by symbolic variables that are used to codify cumulative operations. We show how the

use of macros is helpful in the design of counting strategies on repetitive patterns of a graph.

Macros were first used as a tool during plan execution and analysis in the area of AI. A main property of the macros is the possibility to represent accumulative preconditions and effects by making macros indistinguishable from individual operators, and allowing them to perform efficiently a series of repetitive operations while the same pattern graph is found, as it occurs in the case of polygonal chains. In our case, we analyze the structures lying on polygonal chains P_t, which are generalizations of hexagonal chains, with the purpose of design efficient algorithms for computing $i(P_t)$. Similar strategy is used to compute $i(T_P)$ where T_P is a polygonal tree.

There are several works analyzing extremal values for the Merrifield-Simmons index on hexagonal chain graphs, however to the best of our knowledge, none of those works have presented an efficient algorithm for computing this index.

We present results derived from the application of macros in the determination of numerical properties on graphs whose topologies consist of repetitive 'patterns', such as the case of polygonal arrays and polygonal trees. We focus on the design of an efficient procedure for computing the Merrifield-Simmons index for any kind of polygonal arrays, including polygonal trees formed by polygons.

We show that a polygonal tree T_P has a 2-treewidth decomposition, allowing the application of macros for computing $i(T_P)$ efficiently. Our algorithm could be adapted as a computational tool in the mathematical chemistry area in order to contribute to the analysis of intrinsic properties on molecular graphs. In fact, our algorithm can be adapted to compute also other combinatorial properties on molecular graphs.

This paper is organized as follows. Section 1 presents a brief introduction to the issue of counting independent sets. In section 2, we present the notation to be used. Section 3 shows procedures for counting independent sets on basic topology graphs. Section 4 introduces the computation of Merrifield-Simmon index for polygonal array graphs. Section 5 shows a linear-time procedure for counting independent sets based on a 2-treewidth decomposition of polygonal trees. And finally, some conclusions and final remarks are presented in section 6.

2 Notation

Let $G = (V, E)$ be an undirected graph with vertices set V and set of edges E. The *neighborhood* for $x \in V$ is $N(x) = \{y \in V : xy \in E\}$, and its *closed neighborhood* is $N(x) \cup \{x\}$ which is denoted by $N[x]$. We denote the cardinality of a set A, by $|A|$. The degree of a vertex x, denoted by $\delta(x)$, is $|N(x)|$, and the degree of G is $\Delta(G) = max\{\delta(x) : x \in V\}$.

A path from v to w, denoted as $v - w$ path, is a sequence of the edges: $v_0v_1, v_1v_2, \ldots, v_{n-1}v_n$ such that $v = v_0$, $v_n = w$, and v_k is adjacent to v_{k+1}, for $0 \leq k < n$. The length of the path is n. A simple path is a path where $v_0, v_1, \ldots, v_{n-1}, v_n$ are all distinct. A cycle is a non-empty path such that the first and last vertices are identical, and a simple cycle is a cycle in which no vertex is repeated, with the exception that the first and last vertices are identical. A graph G is acyclic if it has no cycles. P_n, C_n, K_n, denote respectively, a path graph, a simple cycle, and the complete graph, all of those graphs have n vertices.

Given a graph $G = (V, E)$, let $G' = (V', E')$ be a subgraph of G. If E' contains every edge $vw \in E$ where $v \in V'$ and $w \in V'$, then G' is called the *induced graph* of G. A *connected component* of G is a maximal induced subgraph of G, that is, a connected component is not a proper subgraph of any other connected subgraph of G. If an acyclic graph is also connected, then it is called a *tree*. When a vertex is identified as the root of the tree, it is called a *rooted tree*.

The distance $d_G(x; y)$ from a vertex x to another vertex y is the minimum number of edges in a $x - y$ path of G. The distance $d_G(x; S)$ from a vertex x to a set S is the $min_{y \in S} d_G(x; y)$.

Given a graph $G = (V, E)$, $S \subseteq V$ is an independent set of G if for every two vertices v_1, v_2 in S, $v_1v_2 \notin E$. $I(G)$ denotes the set of all independent sets of G. Let $v \in V(G)$, we denote as $I_v(G) = \{S \in I(G) : v \in S\}$ and $I_{-v}(G) = \{S \in I(G) : v \notin S\}$. Our standard reference for graph theoretical terminology is Kocay et al. [11].

Let $G = (V, E)$ be a molecular graph that is, a representation of the structural formula of a chemical compound in tems of graph theory. Denote by $n(G, k)$ the number of ways in which k mutually independent sets can be selected in G. By definition, $n(G, 0) = 1$ and $n(G, 1) = |V(G)|$, for all graphs. $i(G) = \sum_{k \geq 0} n(G, k)$ is called the *Merrifield-Simmons index* of G, that is exactly the number of independent sets of G. $i(G)$ is also called the Fibonacci number of the graph G.

A *matching* of a molecular graph $G = (V, E)$ is a subset $M \subseteq E$ in which any two edges are not incident. A matching M is called a k-matching if $|M| = k$. We denote by $m(G)$ the number of matchings of G, and denote by $m_k(G)$ the number of k-matchings of G. $m(G) = \sum_{k \geq 0} m_k(G)$ is the *Hosoya index* of G, which is exactly the number of edges matching of G.

The corresponding counting problem on independent sets, denoted by $i(G)$, consists of counting the number of independent sets of a graph G. $i(G)$ is a #P-complete problem for graphs G where $\Delta(G) \geq 3$. $i(G)$ remains #P-complete when it is restricted to 3-regular graphs [8].

3 Counting independent sets on basic topology graphs

If G is formed by a set of connected components: $G_i, i = 1, \ldots, k$, then $i(G) = \prod_{i=1}^{k} i(G_i)$, and the time complexity for computing $i(G)$, denoted as $T(i(G))$, is given by the rule of the maximum: $T(i(G)) = max\{T(i(G_i)) : G_i \text{ is a connected component of } G\}$. Thus, one helpful decomposition of the graph is done through its connected components, and from here on, we consider as an input graph only one connected component.

In this section we present two algorithms to compute $i(G)$ for two basic graph patterns: simple cycles and trees. We start by mentioning the relation between the number of independent sets of a path and the Fibonacci numbers.

It turns out that the combinatorial meaning of the Fibonacci numbers are closely related to the number of independent sets of some kind of basic graph patterns. For example, it is shown in [13] that F_{n+2} is equal to the number of subsets (including the empty set) in $\{1, 2, ..., n\}$, such that no two elements are adjacent, i.e. there are not two consecutive integers in any subset. If we think of $\{1, 2, ..., n\}$ as the vertice's set of a path graph, say P_n, where an edge $e_i = \{i, i + 1\},\ i = 1, \ldots, n - 1$ exists for each pair of sequential vertices, then $i(P_n)$ is equal to F_{n+2}, , where F_{n+2} denotes the $n + 2 - th$ Fibonacci number.

3.1 Cycles

If we consider that the n-th element in $\{1, 2, ..., n\}$ is adjacent to the first vertex, then P_n turns into a simple cycle C_n. Let $C_n = (V, E)$ be a simple cycle graph, so $|V| = n = |E| =$

m, i.e. every vertex in V has degree two. We decompose the cycle C_n as: $P_n \cup \{c_m\}$, where $P_n = (V, E')$, $E' = \{c_1, ..., c_{m-1}\}$. P_n is the internal path of the cycle, and $c_m = v_m v_1$ is called the *frond edge* of the cycle.

Let P_n be the internal path of the cycle C_n. Let $\mathcal{F}_i = \{P_i\}, i = 1, \ldots, n$ where $\{P_i\}$ is a family of induced subgraphs $P_i = (V_i, E_i)$ of P_n each built by the set of vertices $\{v_1, \ldots v_i\}$ of V. We associate to each vertex $v_i \in V$ a pair (α_i, β_i). $\alpha_i = |I_{-v_i}(P_i)|$, which means that α_i is the number of subsets in $I(P_i)$ where v_i does not appear. Meanwhile, $\beta_i = |I_{v_i}(P_i)|$ conveys the number of subsets in $I(P_i)$, where v_i appears. Therefore, $i(P_i) = \alpha_i + \beta_i$.

The first pair (α_1, β_1) is $(1, 1)$ since the induced subgraph $P_1 = \{v_1\}$, $I(P_1) = \{\emptyset, \{v_1\}\}$. It is not hard to show (see e.g. De Ita et al. [3]) that a new pair $(\alpha_{i+1}, \beta_{i+1})$ is built from the previous one by a Fibonacci sequence, as it is shown in Equation 1.

$$(\alpha_{i+1}, \beta_{i+1}) = (\alpha_i + \beta_i, \alpha_i) \tag{1}$$

In order to process the number of independent sets on any path of a graph G, we will use *computing threads* or just *threads*. A computing thread is a sequence of pairs $(\alpha_i, \beta_i), i = 1, \ldots, n$ used for computing the number of independent sets on a $v_1 - v_n$ path.

Lemma 1. *The number of independent sets in C_n is equal to $F_{n+1} + F_{n-1}$.*

Proof. Note that every independent set in P_n is an independent set in C_n, except for the sets $S \in I(G)$ where $v_1 \in S$ and $v_n \in S$. In order to eliminate those conflicting sets, we use two computing threads to compute $i(C_n)$. One thread, called the main thread, is used to compute $i(P_n)$, where P_n is the internal path of the cycle, and the other one, the secondary thread, is used to compute $|\{S \in I(P_n) : v_1 \in S \wedge v_n \in S\}|$.

The secondary thread begins with $(\alpha'_1, \beta'_1) = (0, 1)$, in order to consider only the independent sets of $I(P_n)$ where v_1 appears.

By expressing the computation of $i(C_n)$ in terms of Fibonacci numbers and applying recurrence (1), we obtain $(\alpha'_1, \beta'_1) = (0, 1) = (F_0, F_1) \rightarrow (\alpha'_2, \beta'_2) = (1, 0) = (F_1, F_0) \rightarrow (\alpha'_3, \beta'_3) = (1, 1) = (F_2, F_1), \ldots, (\alpha'_n, \beta'_n) = (F_{n-1}, F_{n-2})$, and the value for the final pair is $(0, F_{n-2})$. Therefore, $|\{S \in I(P_n) : v_1 \in S \wedge v_n \in S\}| = 0 + \beta_n = F_{n-2}$. The last pair associated to the computation of $i(C_n)$ is $(F_{n+1}, F_n - F_{n-2}) = (F_{n+1}, F_{n-1})$. Then, $i(C_n) = F_{n+1} + F_{n-1}$, obtaining a well known identity, the n-th Lucas number. □

Thus, the computation of $i(C_n)$ is based on the incremental computation of $i(P_i), i = 1, \ldots, n$. If we perform a linear search on the sequential graph C_n starting at an extreme,

e.g. beginning at v_1, and moving to its incident vertices while the recurrence (1) is applied, then in linear time on the number of vertices (n), we obtain $i(C_n) = \alpha_n + \beta_n = F_{n+1} + F_{n-1}$.

In general, given a connected graph $G = (V, E)$, we call *the charge of v in G*, to the pair (α_v, β_v) associated to a vertex $v \in G$, where $\alpha_v = |I_{-v}(G)|$ and $\beta_v = |I_v(G)|$.

3.2 Counting independent sets on trees

Let $T = (V, E)$ be a rooted tree at a vertex $v_r \in V$ and (α_v, β_v) the charge of $v \in V$. We compute $i(T)$ while traversing T in post-order.

Algorithm 1 Linear_Tree(T)

Input: A tree T
Output: i(T)
Traversing T in post-order, and when a node $v \in T$ is left, assign:
if v is a leaf node in T **then**
$(\alpha_v, \beta_v) = (1, 1)$
else if v is the root node of T **then**
return $\alpha_v + \beta_v$
else if $u_1, u_2, ..., u_k$ are the child nodes of v, as we have already visited all child nodes, then each pair $(\alpha_{u_j}, \beta_{u_j})$ $j = 1, ..., k$ has been determined based on recurrence (1) **then**
Let $\alpha_v = \prod_{j=1}^{k} \alpha_{v_j}$ and $\beta_v = \prod_{j=1}^{k} \beta_{v_j}$.
end if

The Algorithm 1 returns the number of independent sets of a rooted tree T in time $O(n + m)$, which is the necessary time for traversing T in post-order.

The above basic topologies; paths, simple cycles, and trees are graphs where its number of independent sets can be recognized and counted in linear-time. In [3], a polynomial-time algorithm has been presented for computing $i(G)$, when G has a linear compositions of the above patterns, meaning that G is a *cactus graph*, and the algorithm presented is a hybrid procedure between the last two above procedures: the procedure for simple cycles and for trees.

4 Polygonal system graphs

Let C_k be a simple cycle graph of length k. C_k is also called a polygon of size k. A polygonal chain $P_{k,t}$ is a graph obtained by identifying a finite number of t polygons of size at least k, such that each polygon, except the first and the last one, is adjacent to exactly two polygons. When each polygon in $P_{k,t}$ has the same number of k vertices, then

$P_{k,t}$ is a linear array of t k-gons, and is denoted by P_t. In Figure 1 is shown an example of $P_{8,4}$.

The way that two adjacent polygons are joined, via a common vertex or via a common edge, defines different classes of molecular graphs. Let $P_t = h_1 h_2 \cdots h_t$ be a polygonal chain with t polygons, where each h_i and h_{i+1} have exactly one common edge $e_i, i = 1, 2, \ldots, t-1$. A polygonal chain with at least two polygons has two end-polygons: h_1 and h_t. Meanwhile $h_2, \ldots, h_{t-1}$ are the internal polygons of the chain. In a polygonal chain, each vertex has degree either 2 or 3. The vertices of degree 3 are exactly the end points of the common edges between two consecutive polygons.

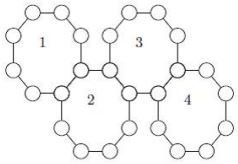

Figure 1. Example of a octagonal zigzag chain

If the array of polygons follows the structure of a tree where instead of nodes we have polygons, and any two consecutive polygons share exactly one edge, then we call to that graph a *polygonal tree* (see the subgraph on the left in Figure 5).

We show in the following section a novel algorithm for computing the Merrifield-Simmons index for any class of polygonal chains, including hexagonal chains. Our algorithm is based on the use of macros for counting the number of independent sets on repetitive structural graphs.

4.1 Counting independent sets on polygonal chains

If we apply the procedure showed in Section 3.1, but using only the symbolic variables: (α_v, β_v) to represent the associated charge for each vertex $v \in C_k$, then we can compute $i(C_k)$ via a macro that is codified by a pair of linear equations on the variables α_v and β_v.

Let us show as an example how to use symbolic variables during the computation of $i(C_4)$. The two computing threads: L_p, L_c, and its associated pairs are expressed as basic operations between the symbolic variables: α and β, in the following way:

$$\begin{array}{lllll} L_p: (\alpha,\beta) \rightarrow & (\alpha+\beta,\alpha) \rightarrow & (2\alpha+\beta,\alpha+\beta) \rightarrow & (3\alpha+2\beta,2\alpha+\beta) \Rightarrow & (3\alpha+2\beta,2\alpha+\beta) \\ L_c: (0,\beta) \rightarrow & (\beta,0) \rightarrow & (\beta,\beta) \rightarrow & (2\beta,\beta) \Rightarrow & \text{- } (0,\beta) \\ & & & & \overline{\qquad\qquad\qquad} \\ & & & & (3\alpha+2\beta,2\alpha) \end{array} \quad (2)$$

The first component of the last pair obtained on the thread L_c is set to 0 since L_c stores only the independent sets were the edge v_1v_4 appears. Thus, the value for $i(C_4)$ is obtained summing the two components of the last pair for L_p and L_c, and under the substitution $\alpha = \beta = 1$. This is, $i(C_4) = 3\alpha + 2\beta + 2\alpha = 5\alpha + 2\beta = 5 + 2 = 7$.

In fact, any $i(C_k)$ can be computed using symbolic variables to express the pair (α, β). Applying the recurrence (1) while each vertex in C_k is visited, as we have shown before, we obtain as the charge of the last vertex $v_k \in C_k$ the pair: $(F_k\alpha + F_{k-1}\beta, F_{k-1}\alpha)$, where F_k is the k-th Fibonacci number. Let us call to the pair $(F_k\alpha + F_{k-1}\beta, F_{k-1}\alpha)$ the macro M_1.

M_1 is a pair, and each one of its components is a linear system of equations defined on the variables: α, β. Furthermore,

$$i(C_k) = (F_k + F_{k-1})\alpha + F_{k-1}\beta = F_{k+1}\alpha + F_{k-1}\beta. \quad (3)$$

The process of forming the pair of linear equations is called the *formation of the macro*. A relevant property of a macro is the possibility to represent cumulative operations via symbolic variables, making macros indistinguishable from individual operators. If subsequences of operators are repeated, a hierarchy of macros can represent a more compactly plan than a simple operator sequence, replacing each occurrence of a repeating subsequence with a macro [1].

M_1 indicates that it does not matter the values for α and β, these variables can be substituted by a current pair of values in order to obtain a final pair of linear equations, which codify the value of $i(C_k)$ applying the recurrence (3).

For example, let $P_2 = h_1h_2$ be two adjacent polygons with common edge xy, as we illustrate in Figure 2. $i(h_1)$ is computed according to procedure of section 3.1, beginning at the common vertex (between both polygons) x and using the symbolic variables: α, β. At the end of the procedure, the macro $M_1 = (F_k\alpha + F_{k-1}\beta, F_{k-1}\alpha)$ is obtained after reaching the other common vertex y. M_1 is associated to the common edge xy between h_1 and h_2.

In our case, the *expansion* of a macro consists in the substitution of the symbolic variables α and β, appearing in its pair of equations, by the current values already computed when the common edge is reached at the computation of $i(h_2)$. This process of expansion is well-defined since no macro appears in its own expansion.

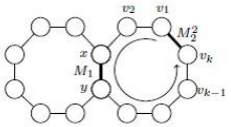

Figure 2. Computing $i(P_2)$

When the computation of $i(h_2)$ is started, for example at vertex v_1 (Fig. 2), two new threads are created $L_P = (\alpha, \beta)$ and $L_c = (0, \beta)$. When the node x is reached, $L_P = (2\alpha + \beta, \alpha + \beta)$ and $L_c = (\beta, \beta)$. When xy is visited, it implies the substitution of α and β in the macro M_1 by the values given at L_P, and L_c pairwise. In our example $L_P = (F_k(2\alpha + \beta) + F_{k-1}(\alpha + \beta), F_{k-1}(2\alpha + \beta))$ and $L_c = (F_k(\beta) + F_{k-1}(\beta), F_{k-1}(\beta))$.

The process of computing the charge on each vertex of h_2 continues with these new pairs of linear equations in both threads. Thus, by applying recurrence (1), the charges are computed until the last vertex v_k of h_2 is reached.

At the end of this process, we obtain a new pair of linear equations that determine the value for the new macro that will be associated to the following common edge if it exists, or it codifies the value for $i(P_t)$. In this way, we can process any polygonal chain P_t in linear time on the number of edges in the array, in fact, in time $O(t \cdot k)$, assuming that there are t k-gons in the array.

In an array of polygons, it is important to define the distance between two consecutive common edges. Therefore, let us consider the counting of adjacent edges in a clockwise direction. Let P_t^r be the polygonal array where there are r edges separating any two consecutive common edges e_i and e_{i+1}. This counting begins from the adjacent edge of e_i and following a clockwise direction on the set of adjacent edges.

For example, let us consider P_3^1 as a polygonal array of three polygons where each pair of consecutive common edges between adjacent polygons are at distance 1. Let us denote as e_1 and e_2 to the two common edges between adjacent polygons. A macro is computed for each common edge $e_i = xy$, starting at vertex x and ending at y.

If we compute the macros starting at e_1, the macro M_1 is associated to the edge e_1, as previously was explained. A macro M_2^1 will be assigned to the edge $e_2 = wx$, where e_2 is at distance one from e_1. This means that $e_2 = wx$ is adjacent to $e = xy$ which is adjacent to $e_1 = yz$. Between e_1 and e_2 there are two vertices.

Therefore, by starting at x until e_1 is reached, we have:

$$L_p:\ (\alpha,\beta) \to (\alpha+\beta,\alpha),\ \text{and}\ L_c:\ (0,\beta) \to (\beta,0).$$

Those last pairs are substituted in the macro M_1 resulting in the new pairs: L_P : $(F_{k+1}\alpha + F_k\beta, F_{k-1}\alpha + F_{k-1}\beta)$ and L_c : $(F_k\beta, F_{k-1}\beta)$. There are $(k-3)$ vertices in P_2^1 before reaching w, the other vertex of e_2. By applying the recurrence (1) to those last pairs, we obtain:

L_p : $(F_{k+1}\alpha + F_k\beta, F_{k-1}\alpha + F_{k-1}\beta) \overset{1}{\to} (F_{k+1}\alpha + F_{k-1}\alpha + F_{k+1}\beta, F_{k+1}\alpha + F_k\beta) \overset{2}{\to}$ $(2F_{k+1}\alpha+F_{k-1}\alpha+F_{k+2}\beta, F_{k+1}\alpha+F_{k-1}\alpha+F_{k+1}\beta) \overset{3}{\to} (3F_{k+1}\alpha+2F_{k-1}\alpha+F_{k+3}\beta, 2F_{k+1}\alpha+$ $F_{k-1}\alpha+F_{k+2}\beta) \ldots \overset{i}{\to} (F_{i+1}F_{k+1}\alpha+F_iF_{k-1}\alpha+F_{k+i}\beta, F_iF_{k+1}\alpha+F_{i-1}F_{k-1}\alpha+F_{k+i-1}\beta)$, and after $(k-3)$ iterations $\ldots \overset{k-3}{\to} (F_{k-2}F_{k+1}\alpha+F_{k-3}F_{k-1}\alpha+F_{2k-3}\beta, F_{k-3}F_{k+1}\alpha+F_{k-4}F_{k-1}\alpha+$ $F_{2k-4}\beta)$.

$L_c : (F_k\beta, F_{k-1}\beta) \overset{1}{\to} (F_{k+1}\beta, F_k\beta) \overset{2}{\to} (F_{k+2}\beta, F_{k+1}\beta) \ldots \overset{k-3}{\to} (F_{2k-3}\beta, F_{2k-4}\beta)$.

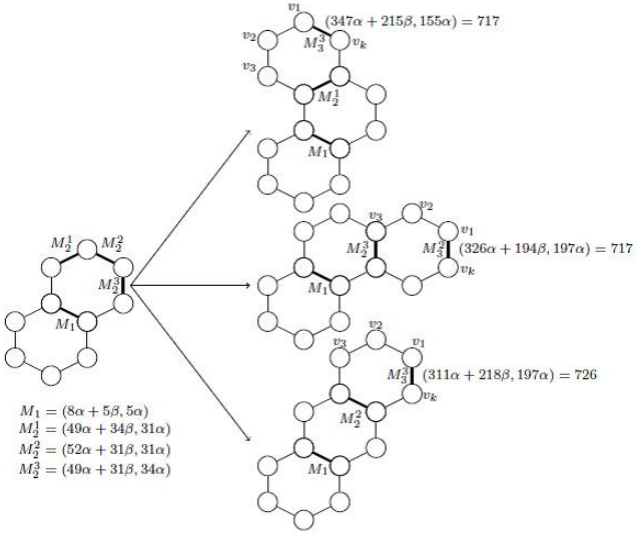

Figure 3. Computing $i(P_3)$ for different distances of their common edges e_2 and e_3

Then, M_2^1 has associated the pair $(F_{k-2}F_{k+1}\alpha + F_{k-3}F_{k-1}\alpha + F_{2k-3}\beta, F_{k-3}F_{k+1}\alpha + F_{k-4}F_{k-1}\alpha + F_{2k-4}\beta) - (0, F_{2k-4}\beta) = (F_{k-2}F_{k+1}\alpha + F_{k-3}F_{k-1}\alpha + F_{2k-3}\beta, F_{k-3}F_{k+1}\alpha + F_{k-4}F_{k-1}\alpha)$.

Notice that $i(P_2^1)$ is obtained summing all values in the macro M_2^1 and assigning $\alpha = \beta = 1$. For example, $i(P_2^1) = (F_{k-2}F_{k+1} + F_{k-3}F_{k-1} + F_{2k-3} + F_{k-3}F_{k+1} + F_{k-4}F_{k-1})$. In Figure 3, we illustrate how the value $i(P_3)$ changes according to the position of the common edge e_2.

In general, for each common edge e_i between two consecutive polygons, a macro M_i is associated to the edge e_i, and such macro can be codified as $(A_i\alpha + B_i\beta, C_i\alpha)$, holding that $i(P_i) = A_i + B_i + C_i$, under the substitution $\alpha = \beta = 1$. Since the cumulative operations for computing $i(P_i)$ are associated as a macro in the common edge of the last polygon of the array, then it can be used to assemble the following polygon in the array. This allows us to do the mathematical analysis when P_t has repetitive patterns.

$i(P_2)$ is invariant to the distance between the common edges e_1 and e_2, but the value $i(P_n)$ changes for $n \geq 3$. A general proof of the above claim has been already demonstrated by Gutman and Zhang [9, 17]. However, their proofs were only for hexagonal chains applying a different method than ours that is based in the application of macros. In our method, we can consider any kind of polygonal arrays.

Notice that $P_{k,t}$ has not restriction on the value of k. The use of macros $M_i = (A_i\alpha + B_i\beta, C_i\alpha)$ allows us to compute $i(P_{k,i}), i = 1, \ldots, t$, and also provides us general recurrences to express $i(P_{k,t})$.

We illustrate how the macros are useful for processing arrays with repetitive structures, and for computing the general recurrences associated with $i(P_t^1)$. In this case, P_t^1 is a polygonal array where any two consecutive common edges e_i and e_{i+1} are at distance 1. And the distance between edges follow a clockwise direction.

Theorem 1. *$i(P_t^1)$, for any $t > 1$, can be computed as $i(P_t^1) = A_t + B_t + C_t$ based on the recurrence:*

$$A_{i+1} = (F_{k-2}(A_i + B_i) + F_{k-3}C_i), \tag{4}$$

$$B_{i+1} = (F_{k-2}A_i + F_{k-3}C_i), \tag{5}$$

$$C_{i+1} = (F_{k-3}(A_i + B_i) + F_{k-4}C_i), \tag{6}$$

with $i = 0, \ldots, t-1$, *and* $A_0 = F_k, B_0 = F_{k-1}, C_0 = F_{k-1}$.

Proof. Let $M_i = (A_i\alpha + B_i\beta, C_i\alpha)$ be the macro computed on P_i^1. If the array is extended by one polygon, the new macro $M_{i+1}^1 = (A_{i+1}\alpha + B_{i+1}\beta, C_{i+1}\alpha)$ is formed, and the new factors $A_{i+1}, B_{i+1}, C_{i+1}$ are expressed based on the previous one A_i, B_i, C_i, in the following way.

Since P_{i+1}^1 has two vertices before finding the macro M_i^1, then:
$L_p: (\alpha, \beta) \to (\alpha + \beta, \alpha)$, and $L_c: (0, \beta) \to (\beta, 0)$.

Those pairs are substituted in the macro M_i^1, resulting in the new pairs: $((A_i + B_i)\alpha + A_i\beta, C_i\alpha + C_i\beta)$ and $(A_i\beta, C_i\beta)$ for L_p and L_c, respectively. Since there are $(k-3)$ vertices in P_{i+1}^1 before finding the following common edge e_{i+1}, then by applying recurrence (1) to those last pairs, we obtain for the thread L_p:

$((A_i + B_i)\alpha + A_i\beta, C_i\alpha + C_i\beta) \overset{1}{\to} ((A_i + B_i + C_i)\alpha + (A_i + C_i)\beta, (A_i + B_i)\alpha + A_i\beta) \overset{2}{\to} ((2(A_i + B_i) + C_i)\alpha + (2A_i + C_i)\beta, (A_i + B_i + C_i)\alpha + (A_i + C_i)\beta) \overset{3}{\to} ((3(A_i + B_i) + 2C_i)\alpha + (3A_i + 2C_i)\beta, (2(A_i + B_i) + C_i)\alpha + (2A_i + C_i)\beta) \overset{j}{\to} ((F_{j+1}(A_i + B_i) + F_iC_i)\alpha + (F_{j+1}A_i + F_jC_i)\beta, (F_j(A_i + B_i) + F_{j-1}C_i)\alpha + (F_jA_i + F_{j-1}C_i)\beta)$, and then after $(k-3)$ iterations, $\ldots \overset{k-3}{\to} ((F_{k-2}(A_i + B_i) + F_{k-3}C_i)\alpha + (F_{k-2}A_i + F_{k-3}C_i)\beta, (F_{k-3}(A_i + B_i) + F_{k-4}C_i)\alpha + (F_{k-3}A_i + F_{k-4}C_i)\beta)$.

Meanwhile, on the second line L_c, we have that $(A_i\beta, C_i\beta) \overset{1}{\to} ((A_i + C_i)\beta, A_i\beta) \overset{2}{\to} ((3A_i + 2C_i)\beta, (2A_i + C_i)\beta) \ldots \overset{k-3}{\to} ((F_{k-2}A_i + F_{k-3}C_i)\beta, (F_{k-3}A_i + F_{k-4}C_i)\beta)$. Since the polygon $i+1$ is a cycle, then the pair $(0, (F_{k-3}A_i + F_{k-4}C_i)\beta)$ is subtracted to the last pair of L_p, forming the new macro $M_{i+1}^1 = ((F_{k-2}(A_i + B_i) + F_{k-3}C_i)\alpha + (F_{k-2}A_i + F_{k-3}C_i)\beta, (F_{k-3}(A_i + B_i) + F_{k-4}C_i)\alpha)$. In this way, we obtain the new factors for the macro M_{i+1}^1. The factor in M_{i+1}^1 can be written in terms of the previous factors of M_i^1 by the recurrences:

$$A_{i+1} = (F_{k-2}(A_i + B_i) + F_{k-3}C_i), \tag{7}$$

$$B_{i+1} = (F_{k-2}A_i + F_{k-3}C_i), \tag{8}$$

$$C_{i+1} = (F_{k-3}(A_i + B_i) + F_{k-4}C_i). \tag{9}$$

On the other hand, the initial macro $M_1 = (F_k\alpha + F_{k-1}\beta, F_{k-1}\alpha)$ is a general macro independent of the size of the first polygon. Then, we obtain from M_1 the initial factors: $A_0 = F_k, B_0 = F_{k-1}$, and $C_0 = F_{k-1}$. □

For example, considering only hexagonal arrays ($k = 6$), M_1 determines the initial

factors: $A_0 = F_k = F_6 = 8$. $B_0 = F_{k-1} = 5$, and $C_0 = F_{k-1} = 5$. Then, $i(P_1^1) = 8 + 5 + 5 = 18$, because $\alpha = \beta = 1$.

When an array of two hexagons is considered, then $A_1 = F_4(A_0 + B_0) + F_3C_0 = 3(8 + 5) + 2 \cdot 5 = 39 + 10 = 49$, $B_1 = (F_4A_0 + F_3C_0) = 3 \cdot 8 + 2 \cdot 5 = 34$, and $C_1 = (F_3(A_0 + B_0) + F_2C_0) = 2 \cdot 13 + 1 \cdot 5 = 31$. Therefore, $i(H_2^1) = 49 + 34 + 31 = 114$.

Similarly, if the array has three hexagons, then $A_2 = F_4(A_1 + B_1) + F_3C_1 = 3(49 + 34) + 2 \cdot 31 = 249 + 62 = 311$, $B_2 = (F_4A_1 + F_3C_1) = 3 \cdot 49 + 2 \cdot 31 = 147 + 62 = 209$, and $C_2 = (F_3(A_1 + B_1) + F_2C_1) = 2 \cdot (49 + 34) + 1 \cdot 31 = 166 + 31 = 197$. And $i(H_3^1) = 311 + 209 + 197 = 717$.

The following lemma shows the recurrence when $r = 2$, the distance between common edges is two. Now, let P_t^2 be the polygonal array where any two consecutive common edges e_i and e_{i+1} are at distance 2. This is, there are two adjacent edges between e_i and e_{i+1}, following a clockwise direction.

Lemma 2. *$i(P_t^2)$, for any $t > 1$, can be computed as $i(P_t^2) = A_t + B_t + C_t$ based on the recurrence:*

$$A_{i+1} = (F_{k-2} + F_{k-5})A_i + F_{k-3}B_i + (F_{k-3} + F_{k-6})C_i, \tag{10}$$

$$B_{i+1} = F_{k-3}(A_i + B_i) + F_{k-4}C_i, \tag{11}$$

$$C_{i+1} = (F_{k-3} + F_{k-6})A_i + F_{k-4}B_i + (F_{k-4} + F_{k-7})C_i, \tag{12}$$

with $i = 0, \ldots, t-1$, and with $A_0 = F_k, B_0 = F_{k-1}, C_0 = F_{k-1}$.

Proof. Let $M_i = (A_i\alpha + B_i\beta, C_i\alpha)$ be the macro computed on P_i^2. If the array is extended by one polygon, the new macro $M_{i+1}^1 = (A_{i+1}\alpha + B_{i+1}\beta, C_{i+1}\alpha)$ is formed. Here, the new factors $A_{i+1}, B_{i+1}, C_{i+1}$ are expressed based on the previous one A_i, B_i, C_i, in the following way.

Since P_{i+1}^2 has three vertices before finding the macro M_i^2, then:
$L_p: (\alpha, \beta) \rightarrow \quad (\alpha + \beta, \alpha) \rightarrow \quad (2\alpha + \beta, \alpha + \beta)$
$L_c: (0, \beta) \rightarrow \quad (\beta, 0) \rightarrow \quad (\beta, \beta)$

Those pairs are substituted in the macro M_i^2, resulting in the new pairs: $((2A_i + B_i)\alpha + (A_i + B_i)\beta, 2C_i\alpha + C_i\beta)$ and $((A_i + B_i)\beta, C_i\beta)$ for L_p and L_c, respectively. And since there are $(k-4)$ vertices in P_{i+1}^1 before finding the following common edge e_{i+1}, then by applying recurrence (1) to those last pairs, we obtain for the thread L_p:

$((2A_i + B_i)\alpha + (A_i + B_i)\beta, 2C_i\alpha + C_i\beta) \xrightarrow{1} ((2A_i + B_i + 2C_i)\alpha + (A_i + B_i + C_i)\beta, (2A_i + B_i)\alpha + (A_i + B_i)\beta) \xrightarrow{2} ((4A_i + 2B_i + 2C_i)\alpha + (2(A_i + B_i) + C_i)\beta, (2A_i + B_i + 2C_i)\alpha + (A_i +$

$B_i + C_i)\beta) \xrightarrow{3} ((6A_i + 3B_i + 4C_i)\alpha + (3(A_i + B_i) + 2C_i)\beta, (4A_i + 2B_i + 2C_i)\alpha + (2(A_i +$ $B_i) + C_i)\beta) \xrightarrow{j} (2 \cdot F_{j+1}A_i + F_{j+1}B_i + 2 \cdot F_jC_i)\alpha + (F_{j+1}(A_i + B_i) + F_jC_i)\beta, (2 \cdot F_jA_i +$ $F_jB_i + 2 \cdot F_{j-1}C_i)\alpha + (F_j(A_i + B_i) + F_{j-1}C_i)\beta)$ and then, after $(k-4)$ iterations, $\ldots \xrightarrow{k-4}$ $(2 \cdot F_{k-3}A_i + F_{k-3}B_i + 2 \cdot F_{k-4}C_i)\alpha + (F_{k-3}(A_i + B_i) + F_{k-4}C_i)\beta, (2 \cdot F_{k-4}A_i + F_{k-4}B_i +$ $2 \cdot F_{k-5}C_i)\alpha + (F_{k-4}(A_i + B_i) + F_{k-5}C_i)\beta)$.

Meanwhile, on the second thread, we obtain after $(k-4)$ iterations, the pair: $((F_{k-3}(A_i + B_i) + F_{k-4}C_i)\beta, (F_{k-4}(A_i + B_i) + F_{k-5}C_i)\beta)$.

Since the polygon $i+1$ is a cycle, then the pair $(0, (F_{k-4}(A_i + B_i) + F_{k-5}C_i)\beta)$ is subtracted to the last pair of L_p, forming the new macro $M^2_{i+1} = (2 \cdot F_{k-3}A_i + F_{k-3}B_i +$ $2 \cdot F_{k-4}C_i)\alpha + (F_{k-3}(A_i + B_i) + F_{k-4}C_i)\beta, (2 \cdot F_{k-4}A_i + F_{k-4}B_i + 2 \cdot F_{k-5}C_i)\alpha)$.

This results in the new factors for the macro M^2_{i+1}. The factor in M^2_{i+1} can be written in terms of the previous factors of M^2_i by the recurrences:

$$A_{i+1} = 2 \cdot F_{k-3}A_i + F_{k-3}B_i + 2 \cdot F_{k-4}C_i, \tag{13}$$

$$B_{i+1} = F_{k-3}(A_i + B_i) + F_{k-4}C_i, \tag{14}$$

$$C_{i+1} = 2 \cdot F_{k-4}A_i + F_{k-4}B_i + 2 \cdot F_{k-5}C_i. \tag{15}$$

And again, the initial macro $M_1 = (F_k\alpha + F_{k-1}\beta, F_{k-1}\alpha)$ determines the initial factors for the previous recurrence, that is, $A_0 = F_k, B_0 = C_0 = F_{k-1}$. □

Similar recurrences can be obtained when we consider different values for the constant r in $i(P^r_t)$, and it holds for any polygonal array P_t without restriction on the number of edges in its polygons.

Notice that for distances $r > 3$ between common edges in the polygonal array, it is necessary to have more than 6 edges forming the polygon. Therefore, if $r = 3 + t, t = 1, 2, \ldots, k-3$, then the polygon has $k \geq 6 + t$ edges.

In fact, for any polygonal array P_t, $i(P_t)$ can be computed traversing by each polygon in the array, even if the distance between consecutive common edges is not a constant. The first polygon determines the initial factors: $A_0 = F_k, B_0 = C_0 = F_{k-1}$. The value for $i(P_{i+1})$ is computed as $A_{i+1} + B_{i+1} + C_{i+1}$, and these factors are computed according to the recurrence determined by the distance r_i between the common edges of P_i and P_{i+1}.

The application of the previous general recurrences provides a linear-time procedure on the number of polygons in the array for computing $i(P_t)$, for any polygonal array.

Thus, the application of macros determines a procedure of order $O(t)$ for computing the Merrefield-Simmons index for any polygonal array P_t.

As an example, let us consider the compound Dibenzoanthracene $P_{6,5}$ that consists in 5 hexagons (see Figure 4), but each consecutive hexagons hold different distances between consecutive common edges. In Table 1, we show how to apply previous recurrence for computing $i(P_{6,5})$. Each macro is generated through the variables A_i, B_i, C_i where $i = 1, \ldots, 5$. Note that in the compound appears alternate distances, r=0 when M_1 is generated. When $r = 1$ we apply the recurrences derived in theorem 1, for $r = 2$ the recurrences from lemma 2. And it is not hard to derive the recurrences when $r = 3$ that we have used in the second column of the Table. In this case, the total number of independent sets is obtained adding $i(P_{6,5}) = A_5 + B_5 + C_5 = 12452 + 8336 + 7889 = 28677$.

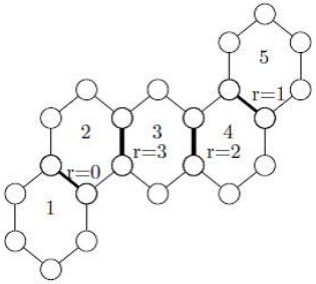

Figure 4. Distances between common edges in the Dibenzoanthracene

Table 1. Computing $i(P_{6,5})$

Var \ P_i	$i=1, r=0$	$i=2, r=3$	$i=3, r=2$	$i=4, r=1$	$i=5, r=1$
	M_1		by ((13,14,15)	by (7,8,9)	by (7,8,9)
A_i	$A_0 = 8$	$A_1 = 49$	$A_2 = 326$	$A_3 = 1954$	$A_4 = 12452$
B_i	$B_0 = 5$	$B_1 = 31$	$B_2 = 194$	$B_3 = 1372$	$B_4 = 8336$
C_i	$C_0 = 5$	$C_1 = 34$	$C_2 = 197$	$C_3 = 1237$	$C_4 = 7889$
$i(P_{6,i})$	18	114	717	4563	28677

In addition, macros can be used to compute $i(G)$ even when G has not repetitive graph topologies, as it is the case for polygonal trees. We show in the following section how to use macros for computing the number of independent sets on polygonal trees.

5 A 2-treewidth decomposition for polygonal trees

Many hard problems can even be solved efficiently on graphs that might not be trees, but are in some sense still sufficiently treelike. A formal parameter that is widely accepted to measure this likeliness is the treewidth of a graph [10].

Treewidth is one of the most basic parameters in graph algorithms. There is a well established theory on the design of polynomial (or even linear) time algorithms for many intractable problems where their input is restricted to graphs of bounded treewidth. More importantly, there are problems on graphs with n vertices and treewidth at most k that can be solved in time $O(c^k \cdot n^{O(1)})$, where c is a problem dependent constant [7].

Algorithm 2 Count_Ind_Sets_Polygonal_trees(G)

Traversing T_P in post-order, and consider all vertex forming the current polygon P_i that is being visited
repeat
1) Form $I_P \subseteq I$ be the indices such that $v \in X(i), i \in I_P$ iff $x \in V(P_i)$
2) Let $X_a, X_b \in X(i), i \in I_P$ be initial nodes of T containing a common edge xy between P_i and its father polygon in T_P. Assume X_a as the father node of X_b
3) Apply algorithm 1 on each $X(i), i \in I_P$ in post-order. The resulting macro is associated to the common edge $xy \in X_a$
/* If there is a macro (α_i, β_i) in any $X(i), i \in I_P$ then a macro-expansion is performed */
4) Eliminate $X(i), i \in I_P$ from T, with exception of X_a
until the root node in T_G is evaluated
Substitute $\alpha = 1, \beta = 1$ in the last macro obtained, resulting a pair of integers (a, b)
Returns $i(T_P) = a + b$.

For example, a maximum independent set (a MIS) of a graph can be found in time $O(2^k \cdot n)$, given a tree decomposition of width at most k. Therefore, a quite natural approach to compute $i(G)$ would be to find a treewidth T_G of G, and to determine how to join the partial results on the nodes of T_G. However, for any general graph G, finding its minimum treewidth is a NP-complete problem.

The treewidth decomposition of T_P, when T_P is any polygonal tree graph, is based in the well-known 2-treewidth decomposition of any simple cycle (a polygon), since any two adjacent polygons have just one common edge, then it is enough to join the 2-treewidth decomposition of the two contiguous polygons with the nodes containing the common edge, as it is shown in Figure 5, where the doble edge notation is used to indicate common

edges between consecutive polygons.

Let $T_G = (T, F)$ be the 2-tree decomposition of $T_P = (V, E)$, where $T = (I, F)$ is a tree, and I is an index set. Let X be a function, $X : I \to 2^V$, satisfying the tree constraints of any k-tree decomposition [7]. We refer to $x \in V(G)$ as a vertex and $X(i) \in T$ as nodes of T. A vertex x is associated with a node $i \in I$, or vice versa, whenever $v \in X(i)$. Our treewidth decomposition of T_P keeps the structure of a tree, and then, we can combine algorithm 1 and algorithm 2 for computing $i(T)$. Furthermore, $i(T_P) = i(T)$.

5.1 Processing polygonal trees

Let us consider that we have obtained the 2-treewidth decomposition of the polygonal tree graph T_P, denoted as G_T. Now, in order to compute $i(T_P)$, we apply macros for processing each vertex and common edge in the bags of G_T for each node $N \in G_T$, based in the algorithm 2.

In Figures 5 and 6, we show the 2-treewidth decomposition of a polygonal tree graph, as well as the application of macros for counting independent sets on the nodes of the tree. Notice that every common edge is visited twice. On the first one, a macro is formed, and in the second one, an expansion of the macro is performed. This provides a linear-time algorithm that traverses by all node and edge on the treewidth.

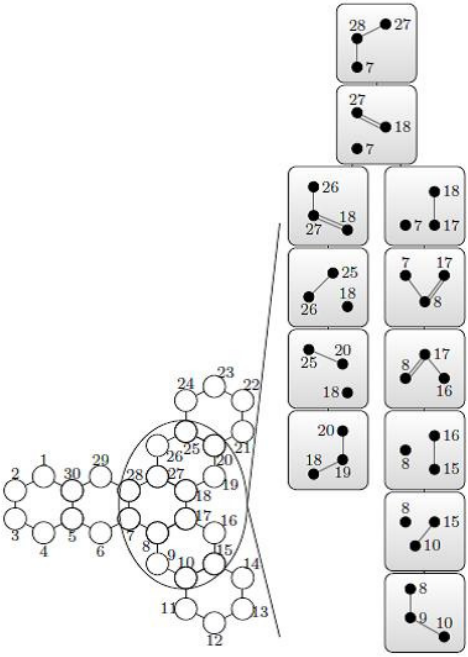

Figure 5. A 2-treewidth decomposition for Trinaftileno

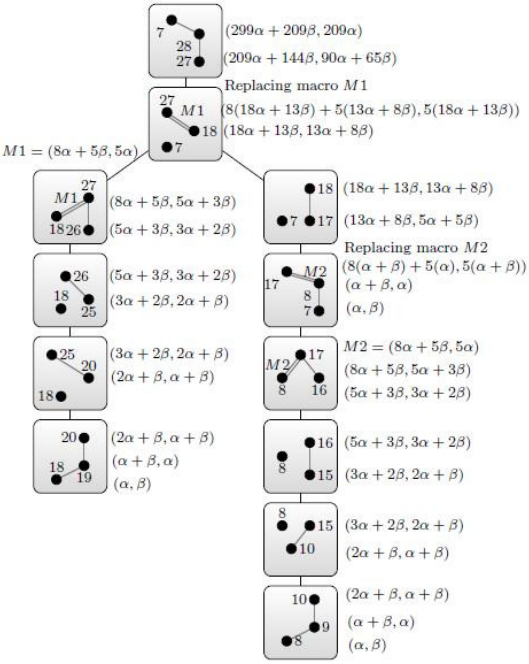

Figure 6. Processing the 2-treewidth decomposition of a polygonal tree

6 Conclusions

One relevant class of graphs used in mathematical chemistry for analyzing the molecular structure in some chemical compounds, is the polygonal chain. We present here a linear-time procedure for computing the Merrefield-Simmon index on this class of graphs. Our procedure applies a common tool used in planning area, which is the use of macros in order to codify cumulative series of basic operations.

In fact, macros provide efficient procedures to count combinatorial objects on polygonal arrays P_n, because P_n has repetitive graphic patterns. In addition, macros can be used for processing a graph G even when G has not repetitive graph topologies, but it has a bounded treewidth decomposition.

We have proposed a 2-treewidth decomposition for any polygonal tree graph T_P, where every common edge between two adjacent polygons will appear exactly in two consecutive nodes of the treewidth. This structure of the treewidth decomposition allows the efficient application of macros for solving counting problems on T_P.

We have presented a novel linear-time algorithm for counting independent sets on T_P. Our algorithm can be adapted to solve other intrinsic properties on polygonal tree graphs, impacting directly on the time complexity of the algorithms for solving these problems.

Acknowledgment: The first two authors thank the support of the program SNI-Conacyt for the accomplishment of this work. We also thank Juan Perdomo Flández who provided a public access to a graphic system, implemented by an exhaustive algorithm, for counting the number of independent sets for graphs in general. Webpage, http://consejos-para-programadores.blogspot.mx/2016/05/.

References

[1] C. Bäckström, A. Jonsson, P. Jonsson, Automaton Plans, *J. Art. Intell. Res.* **51** (2014) 255–291.

[2] Y. Cao, F. Zhang, Extremal polygonal chains on k-matchings, *MATCH Commun. Math. Comput. Chem.* **60** (2008) 217–235.

[3] G. De Ita , A. López-López, A worst–case time upper bound for counting the number of independent sets, in: J. Jeannette, P. Prałat (Eds.), *Combinatorial and Algorithmic Aspects of Networking*, Springer–Veralg, Berlin, 2007, pp. 85–98.

[4] H. Deng, Catacondensed benzenoids and phenylenes with the extremal third–order Randić index, *MATCH Comm. in Math. Comp. Chem.* **64** (2010) 471–496.

[5] H. Deng, The smallest Merrifield–Simmons index of $(n, n+1)$-graphs, *Math. Comput. Modell.* **49** (2009) 320–326.

[6] T. Došlić, F. Måløy, Chain hexagonal cacti: Matchings and independent sets, *Discr. Math.* **310** (2010) 1676–1690.

[7] F. V. Fomin, S. Gaspers, S. Saurabh, A. A. Stepanov, On two techniques of combining branching and treewidth, *Algorithmica* **54** (2009) 181–207.

[8] C. Greenhill, The complexity of counting colourings and independent sets in sparse graphs and hypergraphs, *Comput. Complexity* **9** (2000) 52–72.

[9] I. Gutman, Extremal hexagonal chains, *J. Math. Chem.* **12** (1993) 197–210.

[10] J. Kneis, A. Langer, A practical approach to Courcelles theorem, *El. Notes Theor. Comp. Sci.* **251** (2009) 65–81.

[11] W. Kocay, D. Kreher, *Graphs, Algorithms, and Optimization*, CRC Press, Boca Raton, 2005.

[12] Y. Okamoto, T. Uno, R. Uehara, Counting the number of independent sets in chordal graphs, *J. Discr. Alg.* **6** (2008) 229–242.

[13] H. Prodinger, R. F. Tichy, Fibonacci numbers of graphs, *Fibonacci Quart.* **20** (1982) 16–21.

[14] D. Roth, On the hardness of approximate reasoning, *Art. Intell.* **82** (1996) 273–302.

[15] W. C. Shiu, Extremal Hosoya index and Merrifield–Simmons index of hexagonal spiders, *Discr. Appl. Math.* **156** (2008) 2978–2985.

[16] S. Wagner, I. Gutman, Maxima and minima of the Hosoya index and the Merrifield–Simmons index, *Acta Appl. Math.* **112** (2010) 323–346.

[17] L. Zhang, The proof of Gutman's conjectures concerning extremal hexagonal chains, *J. Sys. Sci. Math. Sci.* **18** (1998) 460–465.

[18] F. Zhang, Z. Li, L. Wang, Hexagonal chains with minimal total π-electron energy, *Chem. Phys. Lett.* **337** (2001) 125-130.

[19] L. Z. Zhang, F. J. Zhang, Extremal hexagonal chains concerning k-matchings and k-independent sets, *J. Math. Chem.* **27** (2000) 319–329.

[20] Y. Zhao, The number of independent sets in a regular graph, *Comb. Prob. Comput.* **19** (2010) 315–320.

[21] Z. Zhu, S. Li, L. Tan, Tricyclic graphs with maximum Merrifield–Simmons index, *Discr. Appl. Math.* **158** (2010) 204–212.