Basic Pattern Graphs for the Efficient Computation of Its Number of Independent Sets

Guillermo De Ita, Miguel Rodríguez(✉), Pedro Bello, and Meliza Contreras

Faculty of Computer Science,
Benemérita Universidad Autónoma de Puebla, Puebla, Mexico
{deita,mrodriguez,pbello,mcontreras}@cs.buap.mx

Abstract. The problem of counting the number of independent sets of a graph G (denoted as $i(G)$) is a classic #P-complete problem. We present some patterns on graphs that allows us the polynomial computation of $i(G)$.

For example, we show that for a graph G where its set of cycles can be arranged as embedded cycles, $i(G)$ can be computed in polynomial time. Particularly, our proposal counts independent sets on outerplanar graphs.

Keywords: Recognition of graph patterns · Counting the number of independent sets · Exact counting

1 Introduction

Counting problems are not only mathematically interesting, but they arise in many applications. For example, if we want to know the probability that a formula in propositional calculus is true, or the probability that a graph remains connected given a probability of failure of an edge, we have to count to approximate such probabilities.

Regarding hard counting problems, the computation of the number of independent sets of a graph has been a key for determining the frontier between efficient counting and intractable counting procedures. Vadhan [8] showed that counting the number of independent sets in graphs of maximum degree 4 is #P-complete. Greenhill [3] refined the previous result showing that counting the number of independent sets on graphs of degree 3 is also #P-complete.

Following the line of exact algorithms, Dahllöf [1] has designed a method for counting independent sets and whose exact algorithm has a worst-case upper bound of $O(1.3247^n), n$ being the number of vertices of the input graph. While Okamoto [5] has shown a linear-time algorithm for counting the number of independent sets for chordal graphs. Efficient algorithms for counting independent

K. M. Figueroa Mora et al. (Eds.): MCPR 2020, LNCS 12088, pp. 57–66, 2020.
https://doi.org/10.1007/978-3-030-49076-8_6

sets have been achieved after to capture structure relations lying in the topology of the graphs, allowing to design special mathematical patterns for counting independent set only on those topologies.

On the other hand, many combinatorial problems ask about embeddings of graphs into other objects [4]. For instance, the polynomial time solvable *graph planarity* problem ask whether a given graph G can be embedded in the plane in such a way that no two edges intersect (except at a common endpoint). In our case, we are interested in a particular subclass of planar graphs, those graphs whose set of vertices can be arranged as incident with the outerface, this class of graphs are called outerplanar graphs. We present here, a novel algorithm for counting the number of independent sets on outerplanar graphs.

2 Notation

Let $G = (V, E)$ be an undirected graph with vertex set V and set of edges E. Two vertices v and w are called *adjacent* if there is an edge $\{v, w\} \in E$, connecting them. Sometimes, the shorthand notation of $u\ v$ is used for denoting the edge $\{u, v\} \in E$.

The *neighborhood* for $x \in V$ is $N(x) = \{y \in V : \{x, y\} \in E\}$ 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 $\Delta(G) = max\{\delta(x) : x \in V\}$. The size of the neighborhood of x, $\delta(N(x))$, is $\delta(N(x)) = \sum_{y \in N(x)} \delta(y)$. A vertex v is *pendant* if $\delta(x) = 1$; and edge $e = \{x, y\}$ is *pendant* if x or y is a pendant vertex.

A path from v to w is a sequence of edges: $v_0v_1, v_1v_2, \ldots, v_{n-1}v_n$ such that $v = v_0$ and $v_n = w$ 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 nonempty path such that the first and last vertices are identical, and a simple cycle is a cycle in which no vertex is repeated, except that the first and last vertices are identical. A graph G is acyclic if it has no cycles. P_n, C_n, R_n, K_n, N_n denote respectively, a path graph, a simple cycle, a start with one center node, the complete graph and the set of n nodes without any edge, all of those graphs have n vertices.

Given a graph $G = (V, E)$, let $G' = (V', E')$ be a subgraph of G if $V' \subseteq V$ and E' contains edges $v, w \in E$ such that $v \in V'$ and $w \in V'$. If E' contains every edge $v, w \in E$ 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. Note that, in a connected component, for every pair of its vertices x, y, there is a path from x to y. If an acyclic graph is also connected, then it is called a *free tree*.

Given a graph $G = (V, E)$, $S \subseteq V$ is an independent set in G if for every two vertices v_1, v_2 in S, $\{v_1, v_2\} \notin E$. Let $I(G)$ denote the set of all independent sets of G. An independent set $S \in I(G)$ is *maximal* if it is not a subset of any larger independent set and, it is *maximum* if it has the largest size among all

independent sets in $I(G)$. The determination of the maximum independent set has received much attention since it is a NP-complete problem.

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 [3]. There are different polynomial procedures for computing $i(G)$ when $\Delta(G) \leq 2$ [1,6,7]. In fact, all of them have linear-time complexity. In the following sections, we present exact combinatorial procedures for computing $i(G)$ according to special patterns existing on the graphs.

3 Basic Graph Patterns for the Efficient Counting of Independent Sets

Since $i(G) = \prod_{i=1}^{k} i(G_i)$ where $G_i, i = 1, \ldots, k$ are the connected components of G [6], then the total time complexity for computing $i(G)$, denoted as $T(i(G))$, is given by the maximum rule as $T(i(G)) = max\{T(i(G_i))$: G_i is a connected component of $G\}$. Thus, a first helpful decomposition of the graph is done via its connected components and from here on, we consider as an input graph only one connected component. We start analyzing the most simple cases for one connected component.

Case A:
Let $P_n = G = (V, E)$ be a graph consisting of a single sequence of nodes (path), i.e. $V = \{1, 2, ..., n\}$ and there exists an edge $e_i = \{i, i+1\}, i = 1, \ldots, n-1$, for each pair of sequential vertices.

We build the family $f_i = \{G_i\}, i = 1, \ldots, n$ where each $G_i = (V_i, E_i)$ is the induced graph of G formed by just the first i vertices of V.

We associate to each vertex $v_i \in V$ a pair (α_i, β_i) where α_i expresses the number of sets in $I(G_i)$ where the vertex v_i does not appear, while β_i conveys the number of sets in $I(G_i)$ where the vertex v_i appears, thus $i(G_i) = \alpha_i + \beta_i$.

The first pair (α_1, β_1) is $(1, 1)$ since for the induced subgraph $G_1 = \{v_1\}$, $I(G_1) = \{\emptyset, \{v_1\}\}$. If we know the value for (α_i, β_i) for any $i < n$, and as the next induced subgraph G_{i+1} is built from G_i adding the vertex v_{i+1} and the edge $\{v_i, v_{i+1}\}$, it is not hard to see that the pair $(\alpha_{i+1}, \beta_{i+1})$ is built from (α_i, β_i) applying the recurrence equation:

$$\alpha_{i+1} = \alpha_i + \beta_i \quad ; \quad \beta_{i+1} = \alpha_i \tag{1}$$

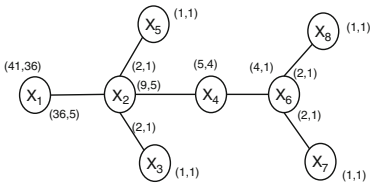
The series (α_i, β_i), i=1,...,n, built from recurrence (1), lead to $i(G_i) = \alpha_i + \beta_i$ for $i = 1, ..., n$. Thus, the computation of $i(G)$ is based on the incremental calculation of $i(G_i), i = 1, \ldots, n$. If we perform a linear search on the sequential graph G starting at an extreme, e.g. beginning at v_1 and moving to its incident vertex while the recurrence (1) is applied, then in linear time on the number of vertices, the formula $i(P_n) = i(G_n) = \alpha_n + \beta_n = F_{n+2}$ is obtained, and where F_n is the $nth-$Fibonacci number.

In order to process the number of independent sets on a path 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 path of n vertices.

Case B:
Let $G = (V, E)$ be a tree. Traversing G in depth first build a rooted tree, whose root node is any vertex $v \in V$, where v was the initial node for beginning the depth first search. We denote with (α_v, β_v) the pair associated with the node v ($v \in G$). We compute $i(G)$ while we are traversing by G in post-order.
Algorithm Count_Ind_Sets_trees(G
Input: G - a tree graph.
Output: The number of independent sets of G
Procedure:
Traversing G in post-order, and when a node $v \in G$ is left, assign:

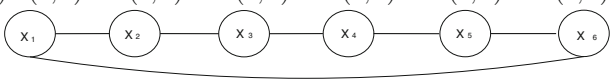
1. $(\alpha_v, \beta_v) = (1, 1)$ if v is a leaf node in G.
2. If v is a parent node with a list of child nodes associated, i.e., $u_1, u_2, ..., u_k$ are the child nodes of v, as we have already visited all child nodes, then each pair $(\alpha_{u_j}, \beta_{u_j})$ $j = 1, ..., k$ has been determined based on recurrence (1). Then, let $\alpha_v = \prod_{j=1}^{k} \alpha_{v_j}$ and $\beta_v = \prod_{j=1}^{k} \beta_{v_j}$. Notice that this step includes the case when v has just one child node.
3. If v is the root node of G then return$(\alpha_v + \beta_v)$.

This procedure returns the number of independent sets of G in time $O(n + m)$ which is the necessary time for traversing G in post-order.



(a) Tree

$(\alpha_i, \beta_i) : (1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3) \rightarrow (8,5) \rightarrow (13,8)$
$(\alpha'_i, \beta'_1) : (0,1) \rightarrow (1,0) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3) \Rightarrow (0,3)$



(b) Cycles

Fig. 1. Counting independent sets over trees and cycles

Example 1. *If* $G = \{(x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_5), (x_4, x_6), (x_6, x_7), (x_6, x_8)\}$ *is a tree, we consider the post-order search and let* x_1 *be the root node of the tree. The number of independent sets at each level of the tree is shown in Fig. 1(a). The procedure Count_Ind_Sets_trees returns for* $\alpha_{x_1} = 41$, $\beta_{x_1} = 36$ *and the total number of independent sets is:* $i(G) = 41 + 36 = 77$.

Case C:
Other basic case is when $G = (V, E)$, $n = m = |V| = |E|$ is a simple cycle, i.e. every vertex in V has degree two. In this case, the cycle can be decomposed as: $G = G' \cup \{c_m\}$, where $G' = (V, E')$, $E' = \{c_1, ..., c_{m-1}\}$. G' is a path of n vertices, and $c_m = \{v_m, v_1\}$ is called as back edge of the simple cycle G.

Observe that every independent set of G is an independent set of G', that is, $I(G) \subseteq I(G')$ since G has one edge more than G'. Thus, if $S \in I(G')$ and $v_1 \in S$ and $v_m \in S$ then S is not an independent set of G. Then, $I(G)$ can be built from $I(G')$ by eliminating those independent sets containing the vertices: v_1 and v_m, that is expressed in the following equation:

$$i(G) = i(G') - |\{S \in I(G') : v_1 \in S \wedge v_m \in S\}| \tag{2}$$

For counting independent sets on a simple cycle, we can use two threads, one of those for computing $i(G')$ and the other thread for computing $|\{S \in I(G') : v_1 \in S \wedge v_m \in S\}|$. This last value can be computed fixing on $I(G')$ the independent sets where v_1 is involved, which is done by computing a thread (α'_i, β'_i), $i = 1,..., m$ where the pair $(\alpha'_1, \beta'_1) = (0, 1)$, considering in this way only the independent sets of $I(G')$ where v_1 appears. We apply (1) for computing the new series: (α'_i, β'_i), $i = 2, \ldots, m$ and also, in order to consider only the independent sets where v_m appears, the final pair (α'_m, β'_m) is taken only as $(0, \beta'_m)$.

In the following examples, we denote with $\rightarrow$ the application of recurrence (1) on (α_i, β_i) in order to obtain $(\alpha_{i+1}, \beta_{i+1})$. And, if we express the new series in terms of Fibonacci numbers, we have that $(\alpha'_1, \beta'_1) = (0, 1) = (F_0, F_1) \rightarrow (\alpha'_2, \beta'_2) = (1, 0) = (F_1, F_0) \rightarrow (\alpha'_3, \beta'_3) = (1, 1) = (F_2, F_1), \ldots, (\alpha'_m, \beta'_m) = (F_{m-1}, F_{m-2})$, and the value for the final pair $(\alpha'_m, \beta'_m) = (0, \beta'_m)$ is $(0, F_{m-2})$, then $|\{S \in I(G') : v_1 \in S \wedge v_m \in S\}| = 0 + \beta_m = F_{m-2}$.

Then, $i(G) = i(G') - |\{S \in I(G') : v_1 \in S \wedge v_m \in S\}| = \alpha_m + \beta_m - \beta'_m = F_{m+2} - F_{m-2}$. Thus, the following theorem is inferred.

Theorem 1. *If* G *is a simple cycle with* n *vertices then the number of independent sets of* G*, expressed in terms of the Fibonacci numbers, is:* $i(G) = F_{n+2} - F_{n-2}$.

Example 2. *Let* $E = \{c_i\}_{i=1}^{6} = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_4\}, \{x_4, x_5\}, \{x_5, x_6\}, \{x_6, x_1\}\}$ *be the set of edges of a simple cycle* $G = (V, E)$*. Let* $G' = (V, E')$ *where* $E = E' \cup \{c_6\}$*, so* G' *is* G *without edge* c_6*. As* G' *is a sequence of 6 vertices then* $i(G') = F_{6+2} = 21$*. While the value for* $|\{S \in I(G') : x_1 \in S \wedge x_6 \in S\}|$ *is* $F_{6-2} = 3$*. Then,* $i(G) = 21 - 3 = 18$ *the computing is shown in Fig. 1(b).*

All the above graph topologies (case A, B and C) represent basic graph patterns that can be recognized and processed to compute its number of independent sets in linear-time. We call *Linear_NI* to the linear procedure that consists of the above three cases (A, B and C). *Linear_NI* will be applied to process any acyclic graph or simple cycles that we find as part of a more complex graph. In fact, in [2] a polynomial-time algorithm has been shown to compute $i(G)$ when G has linear compositions of the above patterns. We can now ask if there exists a family of cyclic connected graphs whose number of independent sets can be computed efficiently, in the next section, we show some families that fulfill this requirement.

4 Recognition of Embedded Cycles

Let $G = (V, E)$ be a connected graph with $n = |V|$, $m = |E|$ and such that $\Delta(G) \geq 2$.

In order to recognize more graph patterns for the efficient computation of $i(G)$, we present the case of the computation of $i(G)$ for outerplanar graphs. For this case, we introduce concepts about the decomposition of a graph by its set of embedded cycles.

If a depth-first search (abbreviated as dfs) is applied over G, starting the search, for example, with the vertex $v_r \in V$ of minimum degree, and selecting among different potential vertices to visit the vertex with minimum degree first and with minimum value in its label as a second criterion, we obtain an unique depth-first graph G' (into the set of all possible depth-first graphs), which we will denote as $G' = dfs(G)$. This dfs also builds an unique spanning tree T_G with v_r as the root node. In time $O(m+n)$, the dfs allows us to detect if G has cycles or not, and the edges forming each cycle. The edges in T_G are called *tree edges*, whereas the edges in $E(G)\backslash E(T_G)$ are called *back edges*. Let $e \in E(G)\backslash E(T_G)$ be a back edge, the union of the path in T_G between the endpoints of e with the edge e itself forms a simple cycle, such cycle is called a basic (or fundamental) cycle of G with respect to T_G. Each back edge $e = \{x, y\}$ holds the maximum path contained in the basic cycle that it is part of. We will call to such maximum path, the *internal path* of a fundamental cycle. Assuming that x is visited first than y during the dfs, we say that x is the start-vertex and y is the end-vertex of the back edge.

According to our particular depth-first search $G' = dfs(G)$ on G, we denote $\mathcal{C} = \{C_1, C_2, ..., C_t\}$ as the set of fundamental cycles found during such depth-first search. Notice that the combination of the procedure for trees and the processing of cycles (Eq. 2) can be applied for computing $i(G)$ if G is a graph where the depth-first search generates a tree and a set of independent fundamental cycles.

If two distinct base cycles C_i and C_j from $\mathcal{C}$ have common edges then we say that both cycles are *intersected*, that is, $C_i \triangle C_j$ form a new cycle, where $\triangle$ denotes the symmetric difference operation between the set of edges in both cycles. In fact, $C_i \triangle C_j = (E(C_i) \cup E(C_j)) - (E(C_i) \cap E(C_j))$ forms a composed

cycle. If two cycles are non-intersected we say that they are *independent*. I.e. two independent cycles (C_i, C_j) hold $(E(C_i) \cap E(C_j)) = \emptyset$. Notice that $t = m - n + 1$ is the dimension of the $\mathbb{Z}_2$-vector space with the symmetric difference on the edge sets as addition, and $\mathcal{C}$ is a base in that $\mathbb{Z}_2$-vector space.

For an outerplanar graph G_o, the cycles in G_o can be considered as embedded cycles, see e.g. Fig. 2. In order to recognize when two cycles C_i and C_j can be expressed as embedded cycles, we use the or-exclusive operation. Given two intersected cycles C_i, C_j, we say that C_i is embedded into C_j, if:
a) $V(C_i) \subset V(C_j)$: the set of vertices of C_i is a subset of the vertices of C_j.
b) $|E(C_i) - E(C_j)| = 1$: there is only one edge from C_i which is not edge of C_j.
c) $C_i \oplus C_j = C_k$: being C_k a new cycle distinct to C_i and C_j and $\oplus$ is the or-exclusive operation between the edges of the cycles.

If the cycles C_i and C_j hold the previous three conditions, we say that C_i is embedded into C_j. Meanwhile, C_j is a cycle that semi-encloses C_i.

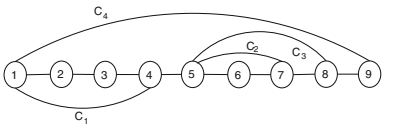
4.1 Processing Outerplanar Graphs

An outerplanar graph G_o could be redrawn in a planar way, where any pair of basic cycles is independent, or one of them is embedded into the other. Thus, G_o is planar, and all its vertices are not enclosed by any edge. Generating a planar drawing is often viewed as a separate problem, in part because drawing algorithms tend to create a planar embedding as a first step, and in part because drawing can be application dependent. In particular, a graph G_o is outerplanar if K_4 and $K_{2,3}$ are forbidden as a minus of G_o. Outerplanar graphs can be recognized in linear-time [9].

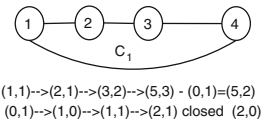
The embedding is a transitive characteristic among embedded cycles. If C_i is embedded into C_j, and C_j is embedded into C_k, then C_i is embedded into C_k. On the other hand, if C_i and C_j are two independent cycles, e.g. $(E(C_i) \cap E(C_j)) = \emptyset$, but there exists a cycle C_k such that C_i and C_j are embedded into C_k, e.g. $(V(C_i) \subset V(C_k))$ and $(V(C_j) \subset V(C_k))$, then we say that (C_k, C_j, C_i) is a tuple of embedded cycles.

A maximal list of embedded cycles $D = (C_1, C_2, \ldots, C_k)$ is a tuple of cycles such that for $i < k$, C_{i+1} is embedded into C_i, $i = 1, \ldots, k-1$, or there exists C_j in the tuple with $j < i \leq k$ such that C_i is embedded into C_j. In a maximal list of embedded cycles $D = (C_1, C_2, \ldots, C_k)$, the cycles are ordered by setting first the most external cycle followed by its internal cycle until arriving to C_k, which is the most internal cycle of the set of embedded cycles. Notice that a maximal list of embedded cycles D is also a graph that we denote by D.

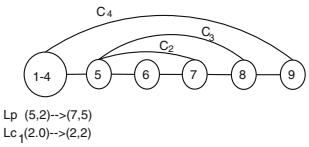
Given a maximal list $D = (C_1, C_2, \ldots, C_k)$ of embedded cycles, the spanning tree of D is called the path of D and it is denoted by P_D. We consider an orientation on P_D; from left to right, or from down to up, according to the drawing of D. The first vertex v_0 of P_D is called the initial vertex of D. Meanwhile, the last vertex v_f of P_D is called the final vertex of D. We will denote as (α_i, β_i) to the pair associated to the vertex $v_i \in V(P_D)$. Given a maximal list of embedded cycles D, we present in this section how to compute $i(D)$.



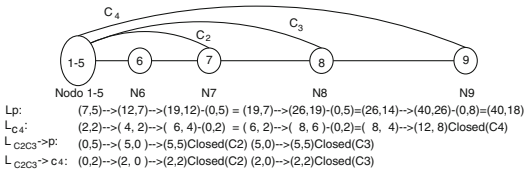
(a) Initial Graph



(b) Processing the first cycle of the list



(c) Processing threads with the contracted node 1-4



(d) Processing threads with the contracted node 1-5

Fig. 2. Computing $i(D)$ with D a maximal list of embedded cycles

Theorem 2. *Given a maximal list of embedded cycles D, $i(D)$ is computed in linear-time on the size of D.*

Proof. We present as proof a linear-time algorithm for the computation of $i(D)$. A main thread, denoted by Lp, is associated to P_D. This thread is always active during all the counting process.

The computation of $i(D)$ is done by traversing in depth-first order the path P_D from its initial node v_0 to its final node v_f. The cycles in D are visited from the most external to the internal cycles according to the depth-first search. Each cycle $C_i, i = 1, 2, \ldots, k$ has a corresponding computing thread L_{C_i}. The computation of $i(C_i)$ follows the case (C) for a simple cycle, described in the previous section. Recurrence (1) is applied on the current pairs $(\alpha_i, \beta_i)_{L_C} \rightarrow (\alpha_i + \beta_i, \alpha_i)_{L_C}$ when a new vertex v_{i+1} of P_D is visited. Each time that an initial vertex of a cycle C_i is visited, the pair $(0, \beta_l)$ is associated to the thread L_{C_i}, where β_l is the value of the second component of the pair (α_l, β_l) associated to Lp, see e.g. Figs. 2(b) and 2(d). When the computation arrives to the end-vertex v of a cycle C_j with corresponding pairs $(\alpha_v, \beta_v)_{C_j}$, then the pair $(0, \beta_v)$ is subtracted to all current computing thread. Afterwards, the computing thread L_{C_j} is closed and stops from being in the computation of $i(D)$. When a cycle C_j has been computed, C_j may be contracted in only one vertex v_{C_j}, and the pair $(\alpha_{C_j}, \beta_{C_j})$, which resulted from the processing of the cycle C_j, is associated to v_{C_j}.

This process continues processing all cycle in D until the depth-first search arrives to the final vertex v_f of D. If (α_f, β_f) is the pair associated to v_f, then $i(D) = \alpha_f + \beta_f$.

We illustrate in the following example, the computation of $i(D)$ when D is a maximal list of embedded cycles.

Example 3. *In Fig. 2(a), we show the input list of embedded cycles D. In Fig. 2(b), as the first cycle of the list is computed, then two computing threads are formed. The path of D is visited in linear way, and at the same time, the recurrence (1) is applied on the current pairs of the computing threads, see Fig. 2(c). Finally, Fig. 2(d) shows the final process of computing on all active threads giving as a result that $i(D) = 40 + 18 = 58$.*

All outerplanar graph can be decomposed in a set of maximal list embedded cycles. For this, let us consider as input to $G_o = (V, E)$ an outerplanar graph. We associate a tree T, called the embedding tree of G_o. Given an embedding tree T of a maximal list of embedded cycles, the final vertex v_f of the list is selected as the root node of T, which make to T a rooted embedding tree. The construction of T satisfies the following properties.

1. The nodes of T are maximal list of embedded cycles.
2. Two nodes v_a, v_b of T are adjacent only if the root vertex of v_a is an internal vertex of the maximal embedding of v_b.
3. For every vertex $v \in V(G_o)$, the subgraph $T_v \subset T$ induced by the maximal embedded cycles containing v is a tree.

Each node v_t of an embedding tree T is formed by a maximal list of embedded cycles, where the final vertex v_f of the list of embedded cycles will be the root node for the subtree $T_{v_t} \subset T$. We show in Fig. 3 a decomposition of an outerplanar graph G_o in its set of maximal embedded cycles, and the formation of the embedding tree T of G_o.

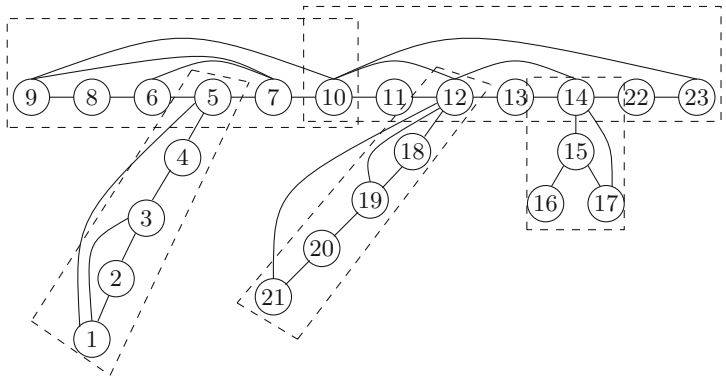


Fig. 3. A decomposition of an outerplanar graph G_o in its set of maximal embedded cycles

Theorem 3. *Let G_o be an outerplanar graph and let T be the embedding tree of G_o, then $i(G_o)$ is computed in polynomial time on the size of G_o.*

Proof. We present as proof a polynomial time algorithm for the computation of $i(G_o)$. A pre-order search on the embedding tree T follows the linear-time procedure developed in the case (B) of the previous section for computing the number of independent sets on tree topologies. Meanwhile, the algorithm presented in Theorem (3) is applied for computing $i(v_t)$ when a node $v_t \in V(T)$ is visited. The combination of both procedures builds a method for the computation of $i(T)$ in polynomial time on the size of the input graph G_o.

5 Conclusions

Computing the number of independent sets of a graph G, denoted as $i(G)$, is a classic #P-complete problem for graphs of degree 3 or higher. We establish that if the depth-first graph of a given graph G has no intersected cycles, then the computation of $i(G)$ is a tractable problem. We have presented a novel algorithm for computing $i(G)$ for any outerplanar graph G.

Our proposal for computing $i(G)$ do not impose restrictions on the degree of the graph, but rather, it depends on its topological structure. Those previous cases allows to establish a finer border between the classes FP and #P for the problem of counting independent sets. Furthermore, our proposal can be adapted to consider other counting problems.

References

1. Dahllöf, V., Jonsson, P.: An algorithm for counting maximum weighted independent sets and its applications. In: Proceedings of Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 292–298. ACM, San Francisco (2002)
2. De Ita, G., López-López, A.: A worst-case time upper bound for counting the number of independent sets. In: Janssen, J., Prałat, P. (eds.) CAAN 2007. LNCS, vol. 4852, pp. 85–98. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77294-1_9
3. Greenhill, C.: The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. Comput. Complex. **9**(1), 52–72 (2000)
4. Johnson, D.S.: The NP-completeness column: an ongoing guide. J. Algorithms **6**(3), 434–451 (1985)
5. Okamoto, Y., Uno, T., Uehara, R.: Linear-time counting algorithms for independent sets in chordal graphs. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 433–444. Springer, Heidelberg (2005). https://doi.org/10.1007/11604686_38
6. Roth, D.: On the hardness of approximate reasoning. Artif. Intell. **82**(1), 273–302 (1996)
7. Russ, B.: Randomized Algorithms: Approximation. Generation, and Counting. Distinguished Dissertations. Springer, London (2001). https://doi.org/10.1007/978-1-4471-0695-1
8. Vadhan, P.: The complexity of counting in sparse, regular, and planar graphs. SIAM J. Comput. **31**(2), 398–427 (2001)
9. Wiegers, M.: Recognizing outerplanar graphs in linear time. In: Tinhofer, G., Schmidt, G. (eds.) WG 1986. LNCS, vol. 246, pp. 165–176. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-17218-1_57