Applying Fibonacci Recurrences for Counting Combinatorial Objects in Graphs

Guillermo De Ita, Meliza Contreras, Pedro Bello

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

Abstract. The main objective of this research is determine a graph-topological property which allows us to count, in polynomial time over the size of a graph, its number of independent sets, edge covers and vertex covers.
We show how to apply the Fibonacci recurrence as well as some other binary recurrences for counting some combinatorial objects in a graph. We consider basic topologies of a graph such as paths, cycles and some combination of them. We obtain that for this basic topologies counting independent sets, vertex covers and edge covers holds a simple algebra involving Fibonacci numbers.
Keywords: Combinatorial Algorithms, Counting Edge Covers, Counting Independent Sets, Counting Vertex Covers, Graph Theory.

1. Introduction

Counting has become an important area in mathematics as well as in theoretical computer science, although it has received less attention than decision problems. As a consequence, we know less about the complexity of counting than about the complexity of decision problems. Actually, there are few counting problems in graph theory that can be solved exactly in polynomial time, indeed an important line of research is to determine the class of graphs (or the class of restrictions) in which any counting problem could be solved in polynomial time.

Counting combinatorial objects over graphs has been an interesting and important area of historical research in Mathematics, Physics and Computer Sciences. Counting problems also arise naturally in Artificial Intelligence (AI) research, since various methods used in automatic reasoning are reduced to counting problems. For example, computing the "degree of belief" in propositional logic, estimating the degree of reliability in a Bayesian belief network, generation of explanations to propositional queries, in Bayesian inference, in truth maintenance systems, and for repairing inconsistent databases [2, 10, 13]. The previous problems come from several AI applications such as planning, expert systems, data-mining, approximate reasoning, etc.

The combinatory problem that we address here is to count over three types of objects: independent sets, edge covers and vertex covers. All of these counting problems are #P-complete problems for graphs in general.

Advances in Computer Science and Engineering.
Research in Computing Science 34, 2008, pp. 3-14

Received 22/03/08
Accepted 26/04/08
Final version 03/05/08

One important goal on complexity algorithm theory is to recognize the class of instances where some hard problems, such as the ones we are considering here, become easy problems, i.e. identify the class of graphs where counting independent sets, vertex covers and edge covers can be done in polynomial time. We show here, some efficient procedures for counting such objects over basic topologies of a graph, and how to extend those procedures for considering more complex topologies.

Other methods have been used for counting objects over graphs. For example, in [9] the computational complexity of the maximum independent set problem on graphs is studied. In [8] fast and simple algoritms for counting the number of vertex covers are designed, while in [6, 7] some methods that show that combinatorial optimization problem for determining keyword conflicts is equivalent to the problem of covering the edges of a graph by complete subgraphs, are presented.

2. Preliminaries

Let $G = (V, E)$ be an undirected graph with vertex set (or nodes set) V and set of edges E. Two vertices v and w are called adjacent if there is an edge $\{v, w\} \in E$, joining them. The Neighborhood for $x \in V$ is $N(x) = \{y \in V : \{x, y\} \in E\}$. We denote the cardinality of a set A, by $|A|$.

The degree of x, denoted by $\delta(x)$, is $|N(x)|$, and the degree of G is $\Delta(G) = max\{\delta(x) : x \in V\}$. A vertex v is *pendant* if its neighborhood contains only one vertex; an edge $e = \{u, v\}$ is *pendant* if one of its endpoints is a pendant vertex.

A path from a vertex v to a vertex w in a graph 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 such that $v_0, v_1, \ldots, v_{n-1}, v_n$ are all distinct. A cycle is just 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.

Given a graph $G = (V, E)$, $S = (V', E')$ is 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 S is called the *subgraph of G induced by S* and it is denoted as $G\|S$. We write $G - S$ to denote the graph $G\|(V - V')$ in the same way, $G - v$ denotes the induced subgraph $G\|(V - \{v\})$, and $G - e$ for $e \in E$ is the subgraph of G formed by V and $E - \{e\}$.

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. Notice that, in a connected component, for every pair of its vertices x, y, there is a path from x to y. A tree graph is an acyclic connected graph.

With respect to counting problems, two complexity classes are basic: the class #P which is conformed by counting problems computable in nondeterministic polynomial time, while the class FP is defined as the class of counting problems computable in deterministic polynomial time. That is, a function f belongs to

#P if and only if there is a nondeterministic Turing machine M that runs in polynomial time and such that $f(x)$ equals the number of accepting computation paths of M on input x. Similar definitions correspond with FP with the only difference being that M is a deterministic Turing machine.

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 $I(G)$ be 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 an NP-complete problem [4].

The corresponding counting problem on independent sets, denoted by $NI(G)$, consists of counting the number of independent sets of a graph G. $NI(G)$ is a #P-complete problem for graphs G where $\Delta(G) \geq 3$. $NI(G)$ remains #P-complete when it is restricted to 3-regular graphs [5].

A *vertex cover* of a graph $G = (V, E)$ is a subset $\mathcal{U} \subseteq V$ that covers every edge of G; that is, every edge has at least one endpoint in $\mathcal{U}$. The counting problem related is to determine the number of vertex covers in G which is denoted by $N\mathcal{U}(G)$. It is known through counting that both independent sets and vertex covers have equivalent complexity time, in fact, one of those problems can be seen as the complement of the other problem [13].

An *edge cover*, $\mathcal{E}$, for a connected graph $G = (V, E)$ is a subset of edges $\mathcal{E} \subseteq E$ which contains edges covering all vertex of G, that is, for each $u \in V$ there is a $v \in V$ such that $e = \{u, v\} \in \mathcal{E}$. We denote by $N\mathcal{E}(G) = |\mathcal{CE}(G)|$ the number of different edge-covers in a graph, and given any graph G we denote the problem of computing the number $N\mathcal{E}(G)$ as the #Edge_Cover problem. It is also common to denote $N\mathcal{E}(G)$ as #Edge_Covers(G).

The problem of counting all edge-covers sets of a graph G, problem denoted as #Edge_Covers(G), is a #P-complete problem via the reduction from #Twice-SAT problem to #Edge_Covers [1]. There is sparse literature about the design of procedures for computing edge covers of a graph, and about of efficient procedures for this problem is even less.

One important subject on complexity algorithm theory is to recognize the class of instances where those counting problems becomes in an easy problem, i.e. identify the class of graphs where counting independent sets, vertex covers and edge covers can be done in polynomial time.

In following sections, we present exact and efficient combinatorial procedures for computing $NI(G)$, as well as $N\mathcal{E}(G)$ and $N\mathcal{U}(G)$ for some classes of graphs.

3. Efficient Counting for Basic Topologies of a Graph

The value $NI(G)$ for any graph G where G could be a disconnected graph, is obtained as the product of $NI(G_i)$, being $G_i, i = 1, \ldots, k$ the set of connected components of G. This product property also occurs when we want to compute

$N\mathcal{E}(G)$ and $N\mathcal{U}(G)$ being G a disconnected graph. Then, we should first determine the set of connected components of G, and this procedure can be done in linear time [12]. Thus, we would consider just the different kinds of connected components in G so, from now on, when we mention a graph G we can suppose that it consists in just one connected component.

The sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ..., in which each number is the sum of the preceding two, is known as the Fibonacci sequence. The numbers in the sequence, known as the Fibonacci numbers, will be denoted by F_i and we formally define them as: $F_0 = 0; F_1 = 1; F_{i+2} = F_{i+1} + F_i, i \geq 0$. Each Fibonacci number can be bounded up and low by $\varphi^{i-2} \geq F_i \geq \varphi^{i-1}, i \geq 1$, and where $\varphi = 1/2(1 + \sqrt{5})$ is known as the 'golden ratio'.

Any Fibonacci number can be computed as $F_i = ClosestInteger(\varphi^i/\sqrt{5})$. Thus, we can compute any Fibonacci number F_i in $O(log(i))$ time.

As general strategy for computing $NI(G), N\mathcal{E}(G)$ and $N\mathcal{U}(G)$ for any graph G, we will use a depth first search over the graph G, since such a search give us an order for visiting each node and edge of G just one time.

We associate a pair (α_i, β_i) to each node of G for counting independent sets and vertex covers, while such pair is associated to each edge for computing edge covers. Thus, a series of pairs: (α_i, β_i) will be computed according each node or edge is visited, and this gives us an incremental strategy for counting objects on any graph.

Processing Independent Sets on Paths

Let $G = P_n(V, E)$ be a linear path (or just a path) with n nodes and $n - 1$ edges. Suppose that we ordered the edges and nodes in P_n in such a way that $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\} = \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}\}$.

We build the family $f_i = \{G_i\}, i = 1, \ldots, n$ where each $G_i = (V_i, E_i)$ is the induced subgraph of G conformed by the first i nodes of V, according to the order given to the elements of V. The method for computing $NI(G)$ is based on the incremental computing of $NI(G_i), i = 1, \ldots, n$. In order to compute $NI(G_i), i = 1, \ldots, n$ we associate to each node v_i a pair (α_i, β_i), where α_i carries the number of independent sets in G_i where the node v_i does not appear while β_i carries the number of independent sets in G_i where the node v_i appears. Then $NI(G_i) = \alpha_i + \beta_i$.

We can start the depth first search at an extreme of the path and moving to its incident nodes until arriving to the other extreme (until arriving to the node v_n). The first pair (α_1, β_1) begins with the value $(1, 1)$ 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 node v_{i+1} and the edge $\{v_i, v_{i+1}\}$, this is, $V_{i+1} = V_i \cup \{v_{i+1}\}, E_{i+1} = E_i \cup \{\{v_i, v_{i+1}\}\}$ then the pair $(\alpha_{i+1}, \beta_{i+1})$ is obtained from (α_i, β_i) applying the recurrence equations (1) on table 1. We will denote with '$\rightarrow$' the application of the recurrence (1) over (α_i, β_i) in order to obtain $(\alpha_{i+1}, \beta_{i+1})$.

For a path P_n, we apply the recurrence (1) according each node is visited, obtaining the series (α_i, β_i) : $(1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3) \rightarrow (8,5) \rightarrow$

Table 1. Recurrence Equations

Symbol	Description	Recurrence	Number
$\rightarrow$	Fibonacci Recurrence	$\alpha_{i+1} = \alpha_i + \beta_i$ $\beta_{i+1} = \alpha_i$	(1)
$\circledcirc$	Covered node and for nodes u, with $\delta(u) > 2$	$\alpha_{i+1} = \alpha_i + \beta_i$ $\beta_{i+1} = \alpha_i + \beta_i$	(2)
$\mapsto$	Visiting a fixed edge	$\alpha_{i+1} = \alpha_i + \beta_i$ $\beta_{i+1} = 0$	(3)
$\curvearrowleft$	Closing Cycle	$\alpha_{i+1} = \alpha_i$ $\beta_{i+1} = \beta_i - \beta'_i$ where β'_i is the computing line over the cycle	(4)

$(13, 8), \ldots$ and this last series coincides with the Fibonacci numbers: $(F_2, F_1) \rightarrow (F_3, F_2) \rightarrow (F_4, F_3) \rightarrow (F_5, F_4) \rightarrow (F_6, F_5) \rightarrow (F_7, F_6)$. Then, we infer that $(\alpha_i, \beta_i) = (F_{i+2}, F_{i+1})$ and then $NI(G) = F_{i+2} + F_{i+1}, i = 1, ..., n$. E.g. for $n = 5$, we have $NI(G_5) = F_7 + F_6 = F_8 = 21$ Thus, the following theorem follows.

Theorem 1 *Let P_n be a path with n nodes, then $NI(P_n) = F_{n+2}$*

Processing Vertex Covers on Paths

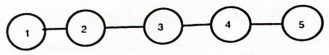
Let P_n be a path of n nodes. A vertex cover C of G is a set of vertices such that for every edge (u, v) of G at least one of u or v is in C. Given a vertex cover C of G and a vertex v in C, we say that v is removable if the set $C - v$ is still a vertex cover of G.

It is NP-hard to determine, given a graph and an integer k, whether the graph has a vertex cover of size at most k [4].

Vertex cover and independent set are very closely related to graph problems. Since every edge in E is incident on a vertex in a cover S, there can be no edge for which both endpoints are not in S. Thus $V - S$ must be an independent set. Further, since minimizing S is the same as maximizing $V - S$, a minimum vertex cover defines a maximum independent set, and vice versa. This equivalence means that if you have a program that solves the independent set, you can use it on your vertex cover problem. For example in Figure 1 we have $S = \{s_1, s_2, \ldots, s_n\}$ where s_i is an independent set and $V = \{v_1, v_2, \ldots, v_n\}$ a set of vertices, for each subset in S applying its complement $S - V$ we obtain a vertex cover.

Corollary 1 *A set S is a vertex cover iff its complement $V - S$ is an independent set in the same graph.*

Using the definition of independent set and vertex cover, the proof is relatively easy. If S is an independent set, then for each edge, at most one endpoint is in S. Thus, for each edge, at least one endpoint is in the complement of S



$V =$	$\{1,2,3,4,5\}$
$S =$	$\{\epsilon,\{1\},\{2\},\{3\},\{4\},\{5\},\{1,3\},\{1,4\},\{1,5\},\{2,4\},\{2,5\},\{3,5\},\{1,3,5\}$
$V-S =$	$\{\{1,2,3,4,5\},\{2,3,4,5\},\{1,3,4,5\},\{1,2,4,5\},\{1,2,3,5\},\{1,2,3,4\},$ $\{2,4,5\},\{2,3,5\},\{2,3,4\},\{1,3,5\},\{1,3,4\},\{1,2,4\},\{2,4\}$

Fig. 1. Obtaining vertex covers

since there are exactly two endpoints for each edge. This means exactly that the complement of S is a vertex cover.

The other direction is similar: if S is a vertex cover, then each edge has at least one endpoint in S, so no edge has both endpoint in the complement of S, and the complement is an independent set.

The directly following conclusion is that the largest size of independent set plus the minimal size of vertex cover in the same graph is the total number of graph nodes n.

Thus, applying the Fibonacci series using the Recurrence (1) in Figure 2, we obtain the value for $N\mathcal{U}(G)$ as $F_{i+2} + F_{i+1}, i = 1, ..., n$. E.g. for $n = 5$, we have $N\mathcal{U}(G) = F_7 + F_6 = F_8 = 21$. Thus, the following theorem follows.

Theorem 2 *Let P_n be a path with n nodes, then $N\mathcal{U}(P_n) = F_{n+2}$*

Processing Edge Covers on Paths

Contrary to the independent sets and vertex cover, to finding an edge cover of maximum size has polynomial time complexity [4], although its counting version is a #P-complete problem.

For counting edge covers $N\mathcal{E}(P_n)$ in a path, we will again apply a depth first search over P_n. During the construction of an edge cover $\mathcal{E}$ of a graph $G = (V, E)$, we distinguish between two different states of a node. We say that a node $u \in V$ is *free* when it has not still been covered by any edge of $\mathcal{E}$, while if the node has already been covered we say that the node is *cover*.

The edges which appear in all edge cover sets are called *fixed* edges. For example, notice that the pendant edges are fixed edges. In the path P_n there are two fixed edges: e_1 and e_{n-1} which they coincide with the extreme edges on the path and with the property that such edges appear in all edge cover sets of P_n.

We associate with each edge $e_i \in P_n, i = 0, \ldots, n-1$ an ordered pair: (α_i, β_i) of integer numbers where α_i expresses the number of sets in $\mathcal{CE}(G_{i+1})$ where the edge e_i appears, while β_i conveys the number of sets in $\mathcal{CE}(G_{i+1})$ where the edge e_i does not appear. Thus, $N\mathcal{E}(G_{i+1}) = \gamma_i = \alpha_i + \beta_i$.

If we visit P_n in depth-first search computing (α_i, β_i) according each edge $e_i = \{v_i, v_{i+1}\}, i = 0, \ldots, n-1$ is visited, we obtain at the end of this traversing a last pair $(\alpha_{n-1}, \beta_{n-1})$ which represent the value for #Edge_Covers(P_n) $= \gamma_{n-1} = \alpha_{n-1} + \beta_{n-1}$.

We have to assign to the first pair (α_0, β_0) of the path the value (1,0) since it means that the edge e_0 always appears in all edge cover sets, notice that for each pendant edge e_p which starts a series of values (α_p, β_p) the initial value has to be $(1, 0)$.

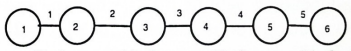
If we know the value for (α_i, β_i) for any $i < n - 2$, and as the next induced subgraph G_{i+1} is built from G_i adding the node v_{i+1} and the edge $e_i = \{v_i, v_{i+1}\}$. Considering that the node v_i is free, for any edge cover set of G_{i+1} where the edge e_{i-1} appears (α_{i-1} cases) the edge e_i can appear or it can not appear since in both cases the node v_i has been covered for e_{i-1}, while for any edge cover set where the edge e_{i-1} does not appear (β_{i-1} cases) the edge e_i must be considered in order to cover the node v_i. So, the recurrence relation for computing the new pair (α_i, β_i) is built from recurrence (1).

The other case is when the node v_i has already been covered, for example when we are considering a node of degree greater than two. In such case, the edge e_i can be taken into account or not since in both cases the node v_i has already been covered, then the computing of the new pair (α_i, β_i) is given by the recurrence (2). We will denote with '$\circledcirc$' the application of the recurrence (2) over (α_i, β_i) in order to obtain $(\alpha_{i+1}, \beta_{i+1})$.

Those two relations give us a complete way to compute the pair (α_i, β_i) in base on the previous pair $(\alpha_{i-1}, \beta_{i-1})$ and considering when the node v_i is free or cover. After to compute (α_i, β_i) we label to the node v_i as a covered node.

When we arrive to the last edge e_{n-1} which is a fixed edge, we have computed the pair $(\alpha_{n-2}, \beta_{n-2})$ and as e_{n-1} has to appear in all edge cover sets of P_n then the edge e_{n-1} appears in all the cases, that is, in $\alpha_{n-1} = \alpha_{n-2} + \beta_{n-2}$ cases, and e_{n-1} has not chance to do not appear in the edge cover sets of P_n, so that $\beta_{n-1} = 0$. This is the case of processing a fixed edge, so we use the recurrence (3) on table 1.

Then, the pair associated with the last edge in the path, is: $(\alpha_{n-1}, \beta_{n-1}) = (\alpha_{n-2} + \beta_{n-2}, 0)$. We will denote with '$\mapsto$' the application of the recurrence (3) over $(\alpha_{n-2}, \beta_{n-2})$ in order to obtain $(\alpha_{n-1}, \beta_{n-1})$.



$Nodes:$	$Node_1$	$Node_2$	$Node_3$	$Node_4$	$Node_5$	$Node_6$
$(\alpha_i, \beta_i):$	$(\alpha_1, \beta_1) \rightarrow$	$(\alpha_2, \beta_2) \rightarrow$	$(\alpha_3, \beta_3) \rightarrow$	$(\alpha_4, \beta_4) \rightarrow$	$(\alpha_5, \beta_5) \rightarrow$	(α_6, β_6)
$NI(G):$	$(1,1) \rightarrow$	$(2,1) \rightarrow$	$(3,2) \rightarrow$	$(5,3) \rightarrow$	$(8,5) \rightarrow$	$(13,8) = 21$
$NU(G):$	$(1,1) \rightarrow$	$(2,1) \rightarrow$	$(3,2) \rightarrow$	$(5,3) \rightarrow$	$(8,5) \rightarrow$	$(13,8) = 21$
$Edges:$		$Edge_1$	$Edge_2$	$Edge_3$	$Edge_4$	$Edge_5$
$NE(G):$		$(1,0) \rightarrow$	$(1,1) \rightarrow$	$(2,1) \rightarrow$	$(3,2) \mapsto$	$(5,0) = 5$

Fig. 2. Processing a linear path

The series $(\alpha_i, \beta_i), i = 1, ..., n - 1$ is built based on recurrence (1), (2) and (3) allow us to compute $N\mathcal{E}(G)$. In Figure 2 we show the way of computing the number of independent sets, the vertex covers and edge covers according to re-

currences in table 1. Starting at an extreme of the path e.g. beginning at v_1 or e_1 and moving to its incident node (edge) while the recurrence (1), (2) or (3) are applied, in linear time over the size of P_n,we obtain the values for $NI(P_6)$, $NU(P_6)$ and $N\mathcal{E}(P_6)$. Then, #Edge_Covers(P_n) is computed as $N\mathcal{E}(P_n) = \alpha_{n-2} + \beta_{n-2}$. And this last value: $\alpha_{n-2} + \beta_{n-2}$ in this case $\alpha_3 + \beta_3 = 3 + 2 = 5$ corresponds to F_n, being F_n the n-th Fibonacci number, this is $N\mathcal{E}(P_5) = F_5 = 5$. Thus, the following theorem is deduced.

Theorem 3 *Let P_n be a path of n nodes, then: $N\mathcal{E}(P_n) = F_n$*

3.1. Processing simple cycles

In this section we show that for the case of simple cycles C_n with n edges, counting independent sets, vertex covers and edges covers, all of them can also be obtained using Fibonacci recurrences. Furthermore, we show that for this topology the three functions: $NI(C_n), NU(C_n)$ and $N\mathcal{E}(C_n)$ coincide in the value: $F_{n+2} - F_{n-2}$.

Processing Independent Sets on Simple Cycles

Let $G = (V, E)$, $|V| = n = |E| = m$ be a simple cycle, i.e. every node 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' constitutes so a path and $c_m = \{v_m, v_1\}$ is the edge which if is added to G', forms the cycle G.

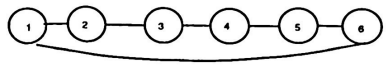
Observe that every independent set S of G is an independent set of G', that is, $S(G) \subseteq S(G')$ since G has one edge more than G'. Thus, if $S \in S(G')$ and $v_1 \in S$ and $v_m \in S$ then S is not an independent set of G. Then, $S(G)$ can be built from $S(G')$ by eliminating those independent sets containing the nodes: v_1 and v_m. Thus, $NI(G) = NI(G') - |\{S \in I(G') : v_1 \in S \wedge v_m \in S\}|$.

We can apply the previous case for computing $NI(G')$ since G' is a path. And, in order to count $|\{S \in S(G') : v_1 \in S \wedge v_m \in S\}|$, we can fix on $S(G')$ the independent sets where v_1 is involved, which is done by computing a new series (α'_i, β'_i), i=1,...,m starting with the pair $(\alpha'_1, \beta'_1) = (0, 1)$ considering in this way only the independent sets of $S(G')$ where v_1 appears. We apply the recurrence (1) for computing the new series: (α'_i, β'_i), i=2, ...,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 $(0, \beta'_m)$.

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 $|\{S \in I(G') : v_1 \in S \wedge v_m \in S\}| = 0 + \beta_m = F_{m-2}$. Thus, we can formulate the following theorem.

Theorem 4 *If G is a simple cycle with n nodes then the number of independent sets of G, expressed in terms of the Fibonacci numbers, is: $NI(G) = F_{n+2} - F_{n-2}$.*

In the previous figure, we denote with '$\curvearrowleft$' the application of recurrence (4) in order to obtain the last pair (α_m, β_m) which closes the cycle.



$$
\begin{array}{ccccccccc}
(\alpha_1,\beta_1) & \rightarrow & (\alpha_2,\beta_2) & \rightarrow & (\alpha_3,\beta_3) & \rightarrow & (\alpha_4,\beta_4) & \rightarrow & (\alpha_5,\beta_5) \rightarrow (\alpha_6,\beta_6) \\
(1,1) & \rightarrow & (2,1) & \rightarrow & (3,2) & \rightarrow & (5,3) & \rightarrow & (8,5) \rightarrow (13,8) \\
(0,1) & \rightarrow & (1,0) & \rightarrow & (1,1) & \rightarrow & (2,1) & \rightarrow & (3,2) \rightarrow (5,3) \\
 & & & & \Rightarrow (13,8) \curvearrowleft (5,3) = (13,5) & & & &
\end{array}
$$

Fig. 3. Obtaining independent sets, vertex covers and edges covers on a cycle

In figure 3, we have that $(\alpha_m,\beta_m) = (\alpha_6,\beta_6) = (13,8)$, $(0,\beta'_m) = (0,\beta'_6) = (0,3)$, and $(13,8)-(0,3) = (13,5)$. Then, $NI(G) = NI(G')$ - $|\{S \in I(G') : v_1 \in S \wedge v_m \in S\}| = F_{m+2} - F_{m-2} = 21 - 3 = 18$.

Processing Vertex Covers on Simple Cycles

Notice that if $\mathcal{U}$ is a vertex cover, then each edge has at least one endpoint in $\mathcal{U}$, so no edge has both endpoint in the complement of $\mathcal{U}$, and the complement is an independent set and applying the corollary(1): $NI(G) = N\mathcal{U}(G)$. For example let be $V = \{1,2,3,4,5,6\}$ in Figure 3 and let be $S_1 = \{1,5\}$ an independent set then $V - S_1 = \{2,3,4,6\}$ is a vertex cover. For each independent set S_i we obtain the vertex cover: $V - S_i = \mathcal{U}_i$.

Therefore, we have that in this case $NI(C_n) = N\mathcal{U}(C_n)$ and theorem 4 holds for counting the number of vertex covers in a simple cycle C_n. Thus $N\mathcal{U}(C_n) = F_{n+2} - F_{n-2}$

Processing Edge Covers on Simple Cycles

Let $P_{n+2} = (V', E')$ be the path where $V' = V \cup \{v_0, v_{n+1}, v_{n+2}\}$ and if we define the edges: $e'_0 = \{v_0, v_1\}, e'_n = \{v_n, v_{n+1}\}$ and $e'_{n+1} = \{v_{n+1}, v_{n+2}\}$ then $E' = (E - \{e_n\}) \cup \{e'_0, e'_n, e'_{n+1}\}$, that is, $E' = \{e'_0, e_1, e_2, \ldots, e_{n-2}, e_{n-1}, e'_n, e'_{n+1}\}$. Notice that e'_0 and e'_{n+1} are the fixed edges on the path P_{n+2}.

We notice that every edge-cover set of C_n is an edge-cover for P_{n+2} if the edge e_n is changed by the edge e'_n, and adding the fixed edges: e'_0 and e'_{n+1} of P_{n+2}, that is, for each $\mathcal{E} \in \mathcal{CE}(C_n)$ we can build an $\mathcal{E}' \in \mathcal{CE}(P_{n+2})$ changing the edge e_n by e'_n (if it appears in $\mathcal{E}$) and adding the fixed edges e'_0 and e'_{n+1}.

Then $N\mathcal{E}(C_n) < N\mathcal{E}(P_{n+2})$, in fact we can count the difference $N\mathcal{E}(P_{n+2}) - N\mathcal{E}(C_n)$ in exactly way. Each edge-cover of P_{n+2} which does not contain to the edges e_1 and e'_n can not be an edge-cover of C_n. Then, in order to compute $N\mathcal{E}(C_n)$, we can count $N\mathcal{E}(P_{n+2})$ and subtract the number of edge-covers of P_{n+2} where the edges e_1 and e'_n do not appear. We apply theorem 3 for computing $N\mathcal{E}(P_{n+2}) = F_{n+2}$.

And, in order to count $|\{\mathcal{E} \in \mathcal{CE}(P_{n+2}) : e_1 \notin \mathcal{E} \wedge e'_n \notin \mathcal{E}\}|$, we can fix on $\mathcal{CE}(P_{n+2})$ the edge-covers where e_1 does not belong to an edge cover by computing a new series (α'_i,β'_i), i=1,...,n starting with the pair $(\alpha'_1, \beta'_1) = (0,1)$, considering in this way only the edge covers where e_1 does not appear. We apply recurrence (1) for computing the new series: (α'_i,β'_i), i=2, ...,n and also, in order

to consider only the edge covers where the edge e'_n does not appear, the final pair (α'_n, β'_n) is taken only as $(0, \beta'_n)$.

Expressing the new series in terms of Fibonacci numbers, we have that $(\alpha'_1, \beta'_1) = (0,1) = (F_0, F_1) \rightarrow (\alpha'_2, \beta'_2) = (1,0) = (F_1, F_0) \rightarrow (\alpha'_3, \beta'_3) = (1,1) = (F_2, F_1), \ldots, (\alpha'_n, \beta'_n) = (F_{n-1}, F_{n-2})$, and the value for the final pair $(\alpha'_n, \beta'_n) = (0, \beta'_n)$ is $(0, F_{n-2})$. Then $|\{\mathcal{E} \in \mathcal{CE}(G') : e_1 \notin \mathcal{E} \wedge e'_n \notin \mathcal{E}\}| = 0 + \beta_n = F_{n-2}$.

So that, $N\mathcal{E}(C_n) = N\mathcal{E}(P_{n+2}) - |\{\mathcal{E} \in \mathcal{CE}(P_{n+2}) : e_1 \notin \mathcal{E} \wedge e'_n \notin \mathcal{E}\}| = \alpha_n + \beta_n - \beta'_n = F_{n+2} - F_{n-2}$. Thus, we can formulate the following theorem:

Theorem 5 *If C_n is a simple cycle with n nodes then the number of edge cover sets expressed in terms of Fibonacci numbers, is: $N\mathcal{E}(C_n) = F_{n+2} - F_{n-2}$.*

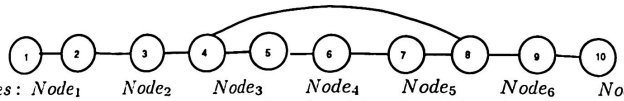
Notice that we are considering that all nodes in the simple cycle C_n are free. But in case that there exists covered nodes in the cycle then we apply the recurrence (2) instead of recurrence (1) when we cross through that covered node.

4. Processing Combinations of Cycles and Paths

If a graph G has some simple combinations of paths and cycles, we can apply the above procedures based on recurrences on table 1 in order to compute $NI(G), N\mathcal{U}(G)$ and $N\mathcal{E}(G)$.

The depth first search is a guide for traversing through the graph and according each node and edge is visited its associated pair (α_i, β_i) is computed. In this case, we consider a main computing thread L_p, that is, a main series of pairs (α_i, β_i). Special care has to be taken at the beginning and the end of any cycle, since the recurrence (4) has to be applied in some cases. We denote by L_c the auxiliary thread for carrying the values to decrement at the end of a cycle.

We show on Figure 4 how to compute $NI(G)$ and $N\mathcal{U}(G)$ for a combination of paths and a cycle. According to this example, we have that $NI(G) = N\mathcal{U}(G) = 132$.



$Nodes:$	$Node_1$	$Node_2$	$Node_3$	$Node_4$	$Node_5$	$Node_6$	$Node_7$
$(\alpha_i, \beta_i):$	(α_1, β_1)	$\rightarrow (\alpha_2, \beta_2)$	$\rightarrow (\alpha_3, \beta_3)$	$\rightarrow (\alpha_4, \beta_4)$	$\rightarrow (\alpha_5, \beta_5)$	$\rightarrow (\alpha_6, \beta_6)$	$\rightarrow (\alpha_7, \beta_7)$
$L_p:$	$(1,1)$	$\rightarrow (2,1)$	$\rightarrow (3,2)$	$\rightarrow (5,3)$	$\rightarrow (8,5)$	$\rightarrow (13,8)$	$\rightarrow (21,13)$
$L_c:$				$(0,3)$	$\rightarrow (3,0)$	$\rightarrow (3,3)$	$\rightarrow (6,3)$
$Nodes:$	$Node_8$		$Node_9$	$Node_{10}$			
$(\alpha_i, \beta_i):$	(α_8, β_8)		(α_9, β_9)	$\rightarrow (\alpha_{10}, \beta_{10})$			
$L_p:$	$(34,21)$	$\curvearrowleft (34,21)$	$\rightarrow (49,34)$	$\rightarrow (83,49) =$	132		
$L_c:$	$(9,6)$	$-(0,6)$					
		$= (34,15)$					

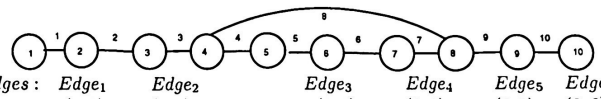
Fig. 4. Counting independent sets and vertex covers on a cycle

In the case of edge covers the procedure is more complex because we should consider covered nodes and fixed edges. If a graph G is a combination of paths and cycles without consider the appearance order of those elements, the method consist in apply recurrence (1) over the principal computing thread L_p. However when a cycle appear, contrary to independent sets and vertex covers, the recurrence (2) is applied over L_p because the actual node has a degree greater than 2.

Let $G = (V, E)$, $|V| = n = |E| = m$ be a path with a simple cycle (see fig. 5). We compute $N\mathcal{E}(G)$ in following way: We initialize the main thread L_p with $(\alpha_i, \beta_i) = (1, 0)$ since e_1 is a fixed edge. The recurrence (1) is applied until edge e_3 where the recurrence (2) has to be applied since $\delta(v_4) > 2$. The edge e_4 generates a new computing thread L_{c1} marking the beginning of the cycle. The initial pair is $(0, 1)$ indicating that the edges e_3 and e_4 do not appear in just one case.

Then, we apply recurrence (1) until arrive to edge e_7 where the recurrence (2) has again to be applied over L_p and L_{c1} because $\delta(v_8) > 2$. At the same time, two new threads are generated $e_7 \rightarrow L_p$ with the values $(0, 9)$ that represents the cases where the edge e_7 does not appear and the thread $e_7 \rightarrow L_{c1}$ with the values $(0, 1)$ that represents the case where the edge e_8 does not appear. The recurrence (2) is applied for each computing thread. As the edge e_8 closes the cycle then the recurrence (4) is applied between the threads L_p with L_{c1} and between $e_7 \rightarrow L_p$ with $e_7 \rightarrow L_{c1}$. The edge e_8 also generates the computing thread L_{e8} with the pair $(0, 8)$ indicating that the edges e_7 and e_8 do not appear in 8 cases. For the edge e_9 the recurrence (2) is applied for two computing threads and then the recurrence (4) is applied in order to close the cycle. Finally, when the computing threads arrive to edge e_{10} we apply the recurrence (1) and after we apply the recurrence (3) for fixed edge in order to obtain $\mathcal{NE} = \alpha_{10} + \beta_{10} = 82$

.



$$
\begin{array}{lllllll}
Edges: & Edge_1 & Edge_2 & & Edge_3 & Edge_4 & Edge_5 \quad Edge_6 \\
L_p: & (1,0) \rightarrow & (1,1) \rightarrow & & (2,1) \circledcirc & (3,3) \rightarrow & (6,3) \rightarrow (9,6) \\
L_{c1}: & & & & & (0,1) \rightarrow & (1,0) \rightarrow (1,1)
\end{array}
$$

$$
\begin{array}{lllll}
Edges: & Edge_7 & Edge_8 & & Edge_9 \\
L_p: & (15,9) \circledcirc & (24,24) \curvearrowleft (3,3) = (24,21) \circledcirc & & (45,45) \curvearrowleft (8,8) = (45,37) \\
L_{c1}: & (2,1) \circledcirc & (3,3) & & \\
e_7 \rightarrow L_p: & (0,9) \circledcirc & (9,9) \curvearrowleft (1,1) = (9,8) & & \\
e_7 \rightarrow L_{c1}: & (0,1) \circledcirc & (1,1) & & \\
L_{e8}: & & & (0,8) \circledcirc & (8,8)
\end{array}
$$

$$
\begin{array}{ll}
Edges: & Edge_{10} \\
L_p: & (45,37) \rightarrow (82,45) \mapsto (82,0) \quad = 82
\end{array}
$$

Fig. 5. Obtaining $N\mathcal{E}(G)$ on a cycle combined with a path

When we consider a tree graph T, we have shown how to compute $NI(T)$ in efficient way [3]. The same algorithm designed for computing $NI(T)$ can be used for computing

$\mathcal{NU}(T)$. Only in the case of computing $\mathcal{NE}(T)$ a more elaborate algorithm has to be designed, even it can work into a polynomial time complexity.

5. Conclusions

We show that the Fibonacci recurrence can be applied for counting; independent sets, vertex covers and edges covers in basic topologies of a graph. For example, for paths, simple cycles and cycles with paths. For this class of topologies, we show that the number of independent sets, the number of vertex covers and the number of edge covers can be computed in polynomial time, in fact, in linear time complexity on the size of the input graph.

The efficient methods can be extended for considering more complex topologies, for example the procedures presented here can be used as a cut's criterion in a branch and bound method for processing general graphs.

References

1. Bubley R., Dyer M., Graph Orientations with No Sink and an Approximation for a Hard Case of #SAT, Proc. of the Eight Annual ACM-SIAM Symp. on Discrete Algorithms, (1997), pp. 248-257.
2. Darwiche Adnan, On the Tractability of Counting Theory Models and its Application to Belief Revision and Truth Maintence, *Jour. of Applied Non-classical Logics*.11 (1-2).(2001), 11-34.
3. De Ita G., López A., A Worst-Case Time Upper Bound for Counting the Number of Independent Sets, Lecture Notes in Computer Science No. 4852, (2007), pp. 85-98.
4. Garey M., Johnson D., Computers and Intractability a Guide to the Theory of NP-Completeness, W.H. Freeman and Co., (1979).
5. Greenhill C., The complexity of counting colourings and independent sets in sparse graphs and hypergraphs", *Computational Complexity*, (2000), 9(1): 52-72 .
6. Grohe Martin., Marx Dániel., Constraint solving via fractional edge covers, *Proceedings of the seventeenth annual ACM-SIAM Symposium on Discrete Algorithms*, (2006), pp.289-298 .
7. Kou L.T., Stockmeyer C.K., Wong C.K., Covering edges by cliques with regard to keyword coflicts and intersection graphs *Communications of the ACM*, Vol. 21, No. 2.(1978) pp.136-139.
8. Lin Min-Sheng., Fast and simple algorithms to count the number of vertex covers in an interval graph, *Information Processing Letters*, Vol. 102, No.4,(2007), pp. 143-146.
9. Lozin V., Milanic., Maximum independent sets in graphs of low degree, *Proceedings of the eighteenth annual ACM-SIAM Symposium on Discrete Algorithms*,(2007) , pp. 874-880.
10. Roth D., On the hardness of approximate reasoning, *Artificial Intelligence 82*, (1996), pp. 273-302.
11. Russ B., *Randomized Algorithms: Approximation, Generation, and Counting*, Distinguished dissertations Springer, (2001).
12. Tarjan R., Depth-First Search and Linear Graph Algorithms, *SIAM Journal on Computing*, Vol. 1, (1972), pp.146-160.
13. Vadhan Salil P., The Complexity of Counting in Sparse, Regular, and Planar Graphs, *SIAM Journal on Computing*, Vol. 31, No.2, (2001), pp. 398-427, 2001.