An Algorithm for Counting the Number of Edge Covers on Acyclic Graphs

J. Raymundo Marcial-Romero¹, Guillermo De Ita², J. A. Hernández¹ and R. M. Valdovinos¹
¹Facultad de Ingeniería, UAEM, Toluca, México
²Facultad de Ciencias de la Computación, BUAP, Puebla, México

Abstract—Counting the number of edge covers on graphs, denoted as the #Edge_Covers problem, is well known to be #P-complete. In this paper, we present an algorithm that compute the number of edge covers in polynomial time if and only if the graph is acyclic. Our algorithm is based on a post-order traversal of the spanning tree of the original graph.

Keywords: Counting the number of edge covers, efficient counting algorithms.

1. Introduction

Counting problems besides of being theoretically interesting, they also have a wide range of applicability on different areas. As a matter of example, it can be mentioned that if a propositional formula needs to be probabilistically tested to be true or given a graph, estimates the probability that it remains connected, in the case of a probability of failure of an edge, the estimation of such probabilities becomes a counting problem. Counting problems also arise naturally in Artificial Intelligence Research, when some methods are used in reasoning areas, such as computing the ‘degree of belief’ and ‘Bayesian belief networks’, which are computationally equivalent to counting the number of models to a propositional formula [4], [5], [11], [13].

Counting has become an important area in theoretical computer science, although it has received less attention than decision problems. 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 low-exponential upper bounds for the time complexity of hard counting problems.

An edge cover set of a graph \( G \) is a subset of edges covering all nodes of \( G \). The problem of counting the number of edge cover sets of a graph, denoted as #Edge_Covers, is a #P complete problem which has been proved via the reduction from #Twice-SAT to #Edge_Covers [1].

Although the time complexity to compute exactly the edge cover sets on a graph is a hard problem, it is relevant to categorize the class of instances where counting the number of edge covers can be done in polynomial time. There is a scarce literature about the design of procedures for computing edge covers, and as far as the authors are aware, it is not known which is the largest polynomial class of graphs for the #Edge_Covers problem.

In this work, the computation of #Edge_Covers based on the topological structure of the graph will be addressed. A method for counting edge covers for acyclic graphs is presented via a post-order traversal strategy.

In Section 2, it is briefly discussed the preliminaries of the paper. In Section 3, the basic topologies of a graph are presented, for which efficient procedures for solving the #Edge_Covers problem have been designed. In this direction, it is shown that the #Edge_Covers problem is solved in linear time over the size of a graph when the graph does not have cycles or it is acyclic. Those are topological cases for which a bound can be estimated from the branch and bound algorithm.

In Section 4, an algorithm to compute edge covers for acyclic graphs is described. A spanning tree of the original algorithm is built followed by a post-order traversal. Finally, in Section 5 the conclusions of the paper are presented.

2. Preliminaries

Let \( G = (V, E) \) be a simple graph (i.e. finite, undirected, loop-less and without multiple edges). \( V(G) \) and \( E(G) \) are also used to denote the set of vertices and edges, respectively, of graph \( G \). A vertex and an incident edge are said to cover each other. The cardinality of a set \( A \) will be as usual denoted by \( |A| \).

The neighbourhood of a vertex \( v \in V \) is the set \( N(v) = \{w \in V : \{v, w\} \in E(G)\} \), and the closure neighbourhood of \( v \) is \( N[v] = N(v) \cup \{v\} \). The degree of a node \( v \), denoted by \( \delta(v) \), is the number of neighbours it contains, that is \( \delta(v) = |N(v)| \). A vertex \( v \) is said to be pendant if its neighbourhood consists of exactly one vertex; analogously an edge \( e \) is said to be pendant if one of its endpoints is a pendant vertex [10]. The degree of a graph \( G \) is \( \Delta(G) = \max_{x \in V(G)} \delta(x) \).

Let \( G = (V, E) \) be a graph then \( S = (V', E') \) is a subgraph of \( G \) if \( V' \subseteq V \) and \( E' \) contains edges \( \{v, w\} \in E \) such that \( v, w \in V' \). If \( E' \) contains every edge \( \{v, w\} \in E \) where \( v, w \in V' \), then \( S \) is called the subgraph of \( G \) induced by \( S \) and is denoted by \( G[S] \). Let \( S \) be any subgraph, \( G - S \) will denote the induced graph \( G[V - V'] \). In the same way, \( G - v \) for \( v \in V(G) \) denotes the induced subgraph \( G[V - \{v\}] \), and \( G - e \) for \( e \in E(G) \) will denote the subgraph of \( G \) formed by \( V(G) \) and \( E(G) - \{e\} \).

Definition 1: Let \( G \) be a graph then \( G \) is said to be connected if for each pair \( v, w \in V(G) \) there exists a path from \( u \) to \( v \). The path may consist of more than one edge \( e \in E(G) \). 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 \( G \).
For example, a tree graph is an acyclic connected graph. Let us denote a complete graph, a simple path and a simple cycle by \( K_n, P_n \) and \( C_n \) respectively, where \( n \) represents the number of nodes in the graph.

**Definition 2:** A vertex cover for a graph \( G = (V, E) \) is a subset of nodes \( U \subseteq V(G) \) that covers every edge of \( G \); that is, every edge has at least one endpoint in \( U \). An edge cover for a graph \( G = (V, E) \) is a subset of edges \( E \subseteq E(G) \) that cover all node of \( G \), that is, for each \( u \in V(G) \) there is a \( v \in V(G) \) such that \( e = \{u, v\} \in E \).

### 2.1 Statement of the problem

The statement of the problem that this paper is concerned about can be established as follows: Let us consider a graph \( G = (V, E) \) and let \( CE(G) = \{E \subseteq E(G) : \mathcal{E} \text{ is an edge cover of } G\} \) be the set of edge covers for \( G \). Let us also consider \( NE(G) = \{|E| \} \) to be the number of edge covers of \( G \), in different words \( NE(G) \) is the cardinality of the set \( CE(G) \). The problem of computing the number \( NE(G) \) for any graph will be called the \#Edge_Covers problem.

### 3. Linear time Procedures for Counting Edge Covers

\( NE(G) \) for any graph \( G \), including the case when \( G \) is a disconnected graph, is computed as: \( NE(G) = \prod_{i=1}^{k} NE(G_i) \), where \( k \) is the cardinality of the set of connected components of \( G \) and each \( G_i \) represents an element of this set. The set of connected components of \( G \) can be computed in linear time [1].

The edges of \( G \) appearing in all edge cover sets are called fixed edges. When an edge cover \( \mathcal{E} \) of \( G \) is being built, we distinguish between two different states of a node \( u \); we say that \( u \) is free when it has not still been covered by any edge of \( \mathcal{E} \), otherwise the node is covered. We begin designing procedures for counting edge covers, considering the most common topologies of a network.

#### Case A: The Bus Topology

Let \( P_n = G = (V, E) \) be a linear bus (a path graph). We assume an order between vertices and edges in \( P_n \), i.e. let \( V = \{v_0, v_1, \ldots, v_n\} \) be the set of \( n + 1 \) vertices and let \( e_i = \{v_{i-1}, v_i\}, 1 \leq i \leq n \) be the \( n \) edges of \( P_n \).

Let \( G_i = (V_i, E_i) \), \( i = 0, \ldots, n \) be the subgraphs induced by the first \( i \) nodes of \( V \), i.e. \( G_0 = (\{v_0\}, \emptyset), G_1 = (\{v_0, v_1\}, \{e_1\}), \ldots, G_n = P_n \). \( G_i, i = 0, \ldots, n \) is the family of induced subgraphs of \( G \) formed by the first \( i \) nodes of \( V \). Let \( CE(G_i) \) be the set of edge covers of each subgraph \( G_i \), \( i = 0, \ldots, n \).

Each edge \( e_i, i = 1, \ldots, n \) in the bus has associated an ordered pair \( (\alpha_i, \beta_i) \) of integer numbers where \( \alpha_i \) carries the number of edge cover sets of \( CE(G_i) \) where the edge \( e_i \) appears in order to cover the node \( v_{i-1} \), while \( \beta_i \) conveys the number of edge cover sets in \( CE(G_i) \) where the edge \( e_i \) does not appear.

By traversing \( P_n \) in depth-first search, each pair \( (\alpha_i, \beta_i), i = 1, \ldots, n \) is computed in accordance with the type of edge that \( e_i \) is. \( P_n \) has two fixed edges: \( e_1 \) and \( e_n \). The pair \( (1,0) \) is assigned to \( (\alpha_1, \beta_1) \) because \( e_1 \) has to appear in all edge cover of \( P_n \).

If we know the values \( (\alpha_{i-1}, \beta_{i-1}) \) for any \( 0 < i < n \), then we know the number of times where the edge \( e_{i-1} \) appears or does not appear into the set of edge covers of \( G_i \). When the edge \( e_i \) is being visited, the vertex \( v_{i-1} \) has to be covered considering its two incident edges: \( e_{i-1} \) and \( e_i \). Any edge cover of \( CE(G_i) \) containing the edge \( e_{i-1} \) (\( \alpha_{i-1} \) cases) has already covered \( v_{i-1} \) then the occurrence of \( e_i \) is optional. But for the edge covers where \( e_{i-1} \) does not appear (\( \beta_{i-1} \) cases) \( e_i \) must appear in order to cover \( v_{i-1} \). This simple analysis shows that the number of edge covers where \( e_i \) appears is \( \alpha_{i-1} + \beta_{i-1} \) and that just in \( \alpha_{i-1} \) edge covers the edge \( e_i \) does not appear. Thus, we compute \( (\alpha_i, \beta_i) \) associated with the edge \( e_i \), applying the Fibonacci recurrence relation.

\[
\alpha_i = \alpha_{i-1} + \beta_{i-1}; \quad \beta_i = \alpha_{i-1}
\]

When the search arrives to the last edge \( e_n \) of the linear bus, we have already computed the pair \( (\alpha_{n-1}, \beta_{n-1}) \); since \( e_n \) is a fixed edge, it has to appear in all edge covers of \( P_n \). We call \( \alpha_n = \alpha_{n-1} + \beta_{n-1} \) and \( \beta_n = 0 \) the recurrence for processing fixed edges (RPF).

The pair associated with \( e_n \) is \( (\alpha_n, \beta_n) = (\alpha_{n-1} + \beta_{n-1}, 0) \). The sum of the elements of this pair \( (\alpha_n, \beta_n) \) yields the number of edge covers: \( NE(P_n) = \alpha_n + \beta_n \). Notice that \( NE(P_n) \) is computed in linear time over the number of edges in \( P_n \). In figure 1 we present an example where \( \rightarrow \) denotes the application of recurrence (1), and \( \Rightarrow \) denotes the application of RPF.

![Fig. 1](image)

**Edges**: \( e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5 \)

\( (\alpha_i, \beta_i) : (1,0) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,0) \)

**Fig. 1**

**Counting edge covers on a linear bus**

Recall that each Fibonacci number \( F_i \) can be bounded from above and from below by \( \phi^{i-2} \geq F_i \geq \phi^{i-1}, i \geq 1 \), where \( \phi = \frac{1}{2} \cdot (1 + \sqrt{5}) \).

**Theorem 3:** The number of edge cover sets of a path of \( n \) edges, is:

\[
F_n = \text{ClosestInteger} \left[ \frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^n \right].
\]

**Proof:** The series \( (\alpha_i, \beta_i), i = 1, \ldots, n \) used for computing \( NE(P_n) \), coincides with the Fibonacci numbers: \( (F_1, F_0) \rightarrow (F_2, F_1) \rightarrow (F_3, F_2) \rightarrow \ldots \rightarrow (F_{n-1}, F_{n-2}) \rightarrow (F_n, 0) \). Then, we infer that \( (\alpha_i, \beta_i) = (F_i, F_{i-1}) \) for \( i = 1, \ldots, n - 1 \) and \( \alpha_n = F_n, \beta_n = 0 \). Thus, \( NE(P_n) = \alpha_n + \beta_n = F_n \).
Case B: The Tree Topology
Let \( T = (V,E) \) be a rooted tree. Root-edges in \( T \) are the edges with one endpoint in the root node; leaf-edges in \( T \) are the edges with one endpoint in a leaf node of \( T \). Given any intermediate node \( v \in V \), we call a child-edge of \( v \) to the edge connecting \( v \) with any of its children nodes, and the edge connecting \( v \) with its father node is called the father-edge of \( v \). \( NE(T) \) is computed by traversing \( T \) in post-order and associating \( (\alpha_e, \beta_e) \) with each edge \( e \in E \), except for the leaf edges.

Algorithm #Edge_Covers_for_Trees(\( T \))
1) Reduce the input tree \( T \) to another tree \( T' \) by pruning all leaf nodes and leaf-edges from \( T \), and by labeling as covered all father nodes of the original leaf nodes of \( T \) (see figure 3).
2) Traverse \( T' \) in post-order and associate a pair \( (\alpha_e, \beta_e) \) with each edge \( e \) in \( T' \). Such pairs are computed in the following way:

- a) \( (\alpha_e, \beta_e) = (1,1) \) if \( e \) is a leaf-edge of \( T' \), since its children nodes have been covered.
- b) if an internal node \( v \) is being visited and it has a set of child-edges \( u_1, u_2, ..., u_k \), then each pair \( (\alpha_{u_j}, \beta_{u_j}), j = 1, ..., k \) has already been computed. Assume \( \alpha_u \) carries the number of different combinations of the child-edges of \( v \) for covering \( v \), while \( \beta_u \) gives the number of combinations among the child-edges of \( v \) which do not cover \( v \). The pair \( (\alpha_u, \beta_u) \), which we assume represents an imaginary child-edge \( e_u \) of \( v \), is computed as:

$$\alpha_u = \prod_{j=1}^{k} (\alpha_{u_j} + \beta_{u_j}) - \prod_{j=1}^{k} \beta_{u_j}; \quad \beta_u = \prod_{j=1}^{k} \beta_{u_j} \quad (2)$$

The pair associated to the father-edge \( e_v \) of \( v \) is computed as:

$$\alpha_v = \begin{cases} \alpha_v + \beta_v, & \text{if } v \text{ is free,} \\ \alpha_v + \beta_v, & \text{otherwise} \end{cases} \quad (\alpha_v, \beta_v) = \begin{cases} (\alpha_v + \beta_v, \alpha_u) & \text{if } v \text{ is free,} \\ (\alpha_v + \beta_v, \alpha_u + \beta_u) & \text{otherwise} \end{cases}$$

This step is iterated until it computes the pairs \( (\alpha_e, \beta_e) \) for all edge \( e \in T' \). If there are more than one root-edges then one extra iteration of this step is applied in order to obtain a final pair \( (\alpha_e, \beta_e) \) associated with just one root-edge \( e_r \).

3) \( NE(T) \) is computed in accordance with the status of the root node \( v_r \) of \( T \); \( NE(T) = \alpha_{e_r} + \beta_{e_r} \) if \( v_r \) is a covered node, otherwise \( NE(T) = \alpha_{e_r} \).

The above procedure returns \( NE(T) \) in time \( O(n + m) \) which is the necessary time for traversing \( T \) in post-order. Notice that this case includes the star topology network.

Example 4: Let \( T \) be the tree of figure 3a. \( T' \) is the reduced tree from \( T \) where its covered nodes are marked by a black point inside of the nodes (figure 3b). When \( T' \) is traversed in post-order a pair \( (\alpha_e, \beta_e) \) is associated with each edge. The pairs for the child-edges of \( v_r \) are: \((1,1),(4,3)\) and \((6,3)\). Those three edges are combined in only one edge \( e_r \) applying recurrence (2): \( \alpha_{e_r} = (1+1)*(4+3)*(6+3)-1*3*3 = 17 \) and \( \beta_{e_r} = 1*3*3 = 9 \). Since \( v_r \) is the root node and it is free, then \( NE(T) = \alpha_{e_r} = 117 \).

Figure 2:

COMPUTING THE NUMBER OF EDGE COVERS FOR A TREE

Case C: The Ring Topology
Let \( C_n = (V,E) \) be a simple ring with \( n \) edges. We assume an order over the nodes and edges of \( C_n \) given by \( V = \{v_1, ..., v_n\} \) and \( E = \{e_i\} \), where \( e_i = \{v_i, v_{i+1}\}, i = 1, ..., n-1, e_n = \{v_n, v_1\} \). We call a computing thread or just a thread to the series \( (\alpha_1, \beta_1) \rightarrow (\alpha_2, \beta_2) \rightarrow ... \rightarrow (\alpha_k, \beta_k) \) obtained by counting in an incremental way, applying the recurrence (1), the number of edge covers of a path with \( k \) edges.

Let \( L_p \) be the thread used for computing the series of pairs associated to the \( n \) edges of \( C_n \). The pair \( (\alpha_1, \beta_1) = (1,1) \) is associated with \( e_1 \) since \( C_n \) has not fixed edges. Traversing in depth first search, the new pairs in \( L_p \) are computed applying the Fibonacci recurrence (1) since all nodes in \( C_n \) have degree two and they are free. After \( n \) applications of recurrence (1), the pair \( (\alpha_n, \beta_n) = (F_{n+1}, F_n) \) is obtained, \( F_i \) being the \( i \)-th Fibonacci number.

Let \( NC_n \) be the number of edge sets counted by \( L_p \), i.e. \( NC_n = \alpha_n + \beta_n = F_{n+2} \). \( L_p \) counted the edge sets where neither \( e_1 \) nor \( e_n \) appear, since \( \beta_1 = 1 \) and \( \beta_n > 0 \). Due to \( e_1 \) or \( e_n \) or both have to be included in the edge cover sets of \( C_n \), in order to cover \( v_1 \), we have to subtract from \( NC_n \) the number of sets which does not cover \( v_1 \).

Let \( Y \) be the number of edge sets which cover all nodes of \( C_n \) except \( v_1 \), then \( NE(C_n) = NC_n - Y \). In order to compute \( Y \) we have to apply the staircase method. \( Y \) is computed by counting edges of the ring \( C_n \), which is a compact form of \( C_n \). The result is associated with the last edge \( e_n \) of the ring \( C_n \) (fig. 4). Notice that the computation of
\(NE(C_n)\) is the order \(O(n)\) since we compute the two threads: \(L_p\) and \(L'_p\) in parallel while the depth-first search is applied.

\[
(\alpha_1, \beta_1) \rightarrow (\alpha_2, \beta_2) \rightarrow (\alpha_3, \beta_3) \rightarrow (\alpha_4, \beta_4) \rightarrow (\alpha_5, \beta_5) \rightarrow (\alpha_6, \beta_6) \\
L_p : (1, 1) \rightarrow (2, 1) \rightarrow (3, 2) \rightarrow (5, 3) \rightarrow (8, 5) \rightarrow (13, 8) \\
C'_6 : (0, 1) \rightarrow (1, 0) \rightarrow (1, 1) \rightarrow (2, 1) \rightarrow (3, 2) \rightarrow (5, 3) \\
\Rightarrow (13, 8) \cap (5, 3) = (13, 5)
\]

**Example 6:** Let \(C_6\) be the ring illustrated in figure 4. Applying theorem (4.3), we have that \(NE(C_6) = F_{6+2} - F_{6-2} = F_8 - F_4 = 21 - 3 = 18\).

The graphs which hold the topologies of the above cases (A) to (C) englobe the most common topologies of a communication network. The linear time procedures designed here can be included into a branch and bound algorithm which processes any kind of topology of a network.

### 4. Counting edge covers for acyclic graphs

In [6] methods to compute the number of edge covers for acyclic graphs and simple cycle graphs were presented. In this section we present an algorithm which combine those methods to compute the number of edge covers for graph without intersecting cycles, e.g. acyclic or with independent cycles graphs. The complexity of the method is polynomial with respect to the number of vertices of the graph.

Let \(G\) be a graph a directed depth first search graph \(T_G\) of \(G\) is built as follows:

1. Built a depth first search graph \(G'\) of \(G\) (it is well known that the edges of \(T_G\) are tree edges or back edges).
2. For each tree edge \(e = (u,v) \in G'\), add the directed edge \(e = u \rightarrow v\) to \(T_G\) if \(u\) is a child of \(v\) in \(G'\).
3. For each back edge \(e = (u,v) \in G'\) add the directed edge \(e = u \rightarrow v\) to \(T_G\) if \(u\) is a descendant of \(v\) in \(G'\).

**Example 7:** Consider the graph of figure 4. It can be notice that it does not have intersecting cycles, just two independent cycles. A depth first search of \(G\) is shown at the left of figure 6. The dotted edges denote back edges and the solid edges denote tree edges. At the right hand side of figure 6 the directed depth first search graph of \(G\) is shown.

**Definition 8:** Let \(T_G\) be a directed depth first search graph and \(v \in V(T_G)\) we define:

\[
\text{input}(u) = \{ v \mid v \rightarrow u \in E(T_G) \} \\
\text{output}(u) = \{ v \mid u \rightarrow v \in E(T_G) \}
\]

**Definition 9:** Let \(G\) be a directed depth first search graph, and \(e = u \rightarrow v \in E(G)\). A tuple \((\alpha_e, \beta_e)\) is associated to \(e\) such that \(\alpha_e\) represents the number of edge covers of \(G\) where \(e\) is considered to cover \(u\). The number \(\beta_e\) represents the edge covers of \(G\) where \(e\) is not considered to cover \(u\).
considered. Line 6 represents the case where the edge $e = u \rightarrow v$ must be present in each edge cover of $T_G$ due to $u$ is a leaf and there is not another edge to cover $u$, so a pair (1, 0) is associated to this kind of edges. Line 9 represents the case where there are two output edges from $u$, let say $e = u \rightarrow v$ and $e_1 = u \rightarrow w$. It can be notice that in a non-intersecting-cycle graph $G$, $\text{output}(v)$ is either 0, 1 or 2 for each node $v \in G$. That $\text{output}(v) = 2$ means that one is a tree edge and the other a back edge (there are not two fathers for a child in a tree). In this case a pair (1, 1) can be associated to $e$ since removing $e$ from an edge cover is valid iff $e_1$ is contained. From line 12 to 15, it is said how to compute $(\alpha_{e_i}, \beta_{e_i})$ when besides that $e = u \rightarrow v$ there is a back edge $e_1 = x \rightarrow v$ (in other words a symple cycle is reached). A detail explanation is shown in [6]. From line 16 to 19 the formula to compute the pair $(\alpha_{e_i}, \beta_{e_i})$ for $e = u \rightarrow v$ taking into account the pairs $(\alpha_{e_j}, \beta_{e_j})$ where $e_j = x_j \rightarrow u$ is described. Finally, lines 20 to 22 is it said how to compute edge covers when the root node is reached. Figure 7 shows the application of algorithm 1 to the directed depth first search graph of figure 6.

### 5. Conclusions

Sound and correct algorithms have been presented to compute the number of edge covers for graphs. It has been shown that if a graph $G$ has simple topologies: paths, trees and simple cycles; then the number of edge covers can be computed in linear time over the graph size.

With respect to cyclic graphs that have intersecting cycles, a branch and bound procedure has been presented, it reduces the number of intersecting cycles until basic graphs are produced.

It has been determined a pair of recurrence relations that establish a bound on the time to compute the number of edge covers on intersecting cycle graphs. It was also designed the first "low-exponential" algorithm for the #EdgeCovers problem whose upper bound in the worst case is $O(1.465571^{(m-n)} \cdot (m+n))$, $m$ and $n$ being the number of edges and nodes of the input graph respectively. In terms only of the number of edges, our algorithm has an upper bound of $O((1.324718)^m \cdot (m+n))$.

### References


