Counting Edge Covers Sets for Estimating the Relevance of Communication Lines

Guillermo De Ita, Meliza Contreras and Pedro Bello

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

Abstract. Counting the number of edge covers on graphs is a #P-complete problem. We design efficient exact methods for computing the number of edge covers for acyclic graphs. Even more, we show that if a graph G does not contain intersectig cycles (any pair of cycles has not common edges) then its number of edge covers can be computed in linear time over the size of the graph G. This determines a border between efficient counting and exponential time procedures for counting the number of edge covers.
We also show how to apply the computing of the number of edge cover for estimating the relevance of the lines in a communication network, which is an important problem in the reliability analysis of a network.
Keywords: Counting Edge Covers, Combinatorial Algorithms, Reliability Analysis Network.

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 propositional formula 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. Counting problems also arise naturally in Artificial Intelligence research. For example, some methods used in reasoning, such as computing 'degree of belief' and 'Bayesian belief networks' are computationally equivalent to counting the number of satisfying assignments to a propositional formula [8, 10]

Counting has become an important area in mathematics as well as in theoretical computer science, although it has received less attention than 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) for which a counting problem could be solved in polynomial time.

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 via the reduction from #Twice-SAT to #Edge_Covers [1].

Although the computation of #Edge_Covers is a hard problem, it is relevant to recognize the class of instances where this problem becomes an easy problem,

Advances in Computer Science and Engineering.
Research in Computing Science 42, 2009, pp. 3-14

Received 14/03/09
Accepted 27/04/09
Final version 05/05/09

that means, to identify the class of graphs where counting the number of its 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 we know, it is not known which is the largest polynomial class of graphs for the #Edge_Covers problem.

We address the computation of #Edge_Covers based on the topological structure of the graph. We focus on determining the topology of the graph G which allows to count the number of edge covers of G in an exactly and efficiently way. We show here that the #Edge_Covers problem can be computed in polynomial time for any acyclic graph.

We show the relevance to compute the number of edge cover for computing the relevance of the lines into a communication network. We show that the computation of the number of edge covers is an important value for estimating the 'strategic' value of each line of a network.

2 Preliminaries

A connected graph is a graph such that there exists a path between all pairs of vertices. If the graph is a directed graph, and there exists a path from each vertex to every other vertex, then it is a strongly connected graph.

A vertex cover of a graph $G = (V, E)$ is a subset $U \subseteq V$ that covers every edge of G; that is, every edge has at least one endpoint in U.

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 $e = \{u, v\} \in \mathcal{E}$.

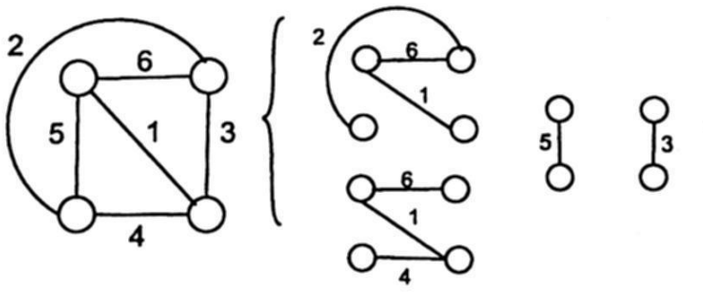


Fig. 1. Cases of edge covers

Given a connected graph $G = (V, E)$, let $C_\epsilon(G) = \{\epsilon \subseteq E : \epsilon$ is an edge cover of $G\}$ be the set of edge-covers sets that a graph G has. Let $NE(G) = |C\epsilon(G)|$ be the number of different edge-covers in a graph, and given any graph G, we denote the problem of computing the number $NE(G)$ as the #Edge_Covers problem.

The #Edge_Covers Counting Problem consists in given a connected graph, find the total number of edge covers of G. For example, in the figure 1 we show three different edge covers sets of a graph.

3 CEC: Counting Edge Covers

The value $NE(G)$ for any graph G, including the case when G is a disconnected graph, is obtained as: $NE(G) = \prod_{i=1}^{k} NE(G_i)$, where $G_i, i = 1, \ldots, k$ is the set of connected components of G. We should first determine the set of connected components of G, and this procedure can be done in linear time. Then, the time complexity of computing $NE(G)$ depends on the maximum time complexity of its connected components. Thus, we would consider just the different kinds of connected components in G, so from now on, when we mention a graph G we suppose that it consists of just one connected component.

We call *fixed edges* to the edges of a graph G that appear in all edge cover set of G. When an edge cover $\mathcal{E}$ of a graph is being built, we distinguish between two different states of a node; we say that a node u 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*.

In order to present the basic procedures for counting edge covers, we consider first the case when the graph is an acyclic graph, and we start analyzing the most simple topology of an acyclic graph. An initial procedure for counting edge covers is to apply a depth first search over the input graph. A basic a recursive procedure scheme for the depth-first search, is the following.

Algorithm 1 Procedure $dfs(G, v)$

```
Mark v as discovered
for each vertex w ∈ N(v) do
  if (w is undiscovered) then
    dfs(G, w)
  end if
  Mark v as finished
end for
```

Case A: Counting Edge Covers on Paths

Let $P_n = G = (V, E)$ be 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. $G_0 = (\{v_0\}, \emptyset), G_1 = (\{v_0, v_1\}, \{e_1\}), G_2 = (\{v_0, v_1, v_2\}, \{e_1, e_2\}), \ldots, G_n = P_n = (V, E)$. $G_i, i = 0, \ldots, n$ is the family of induced subgraphs of G formed by the first i nodes of V. Let $\mathcal{CE}(G_i) = \{\mathcal{E} \subseteq E_i : \mathcal{E}$ is an edge cover of $G_i\}$ be the set of edge covers of each subgraph G_i, $i = 0, \ldots, n$.

We want to count the edge cover sets of P_n by considering the different ways of covering each node on the path. We travel by P_n in depth-first search and when a node is being visited, we consider the different ways to cover it. A path P_n has two special edges: e_1 and e_n which are fixed edges and for $i = 1, \ldots, n-1$, $\delta(v_i) = 2$, e_{i-1} and e_i are the two incident edges of v_i.

We associate with each edge $e_i, i = 1, \ldots, n$ in the path, an ordered pair: (α_i, β_i) of integer numbers where α_i expresses the number of edge cover sets in $\mathcal{CE}(G_i)$ where the edge e_i appears in order to cover the node v_{i-1}, while β_i conveys the number of edge cover sets in $\mathcal{CE}(G_i)$ where the edge e_i does not appear, since v_{i-1} has been already covered (perhaps by the edge e_{i-1}).

Traversing by P_n in depth-first search, each pair (α_i, β_i) is computed in accordance with the type of edge $e_i, i = 1, \ldots, n$ which is being visited. At the end of the search, the last pair (α_n, β_n) is computed, and such pair gives the value for $NE(P_n) = \alpha_n + \beta_n$.

The pair (1,0) is assigned to (α_1, β_1) since the edge e_1 is a fixed edge and e_1 has to appear in all edge cover of P_n. In general, any fixed edge e_p which starts a series (α_p, β_p) has (1,0) as initial pair.

If we know the pair $(\alpha_{i-1}, \beta_{i-1})$ for any $i < n$, 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-1}. When the edge e_i is being visited, the vertex v_{i-1} has to be covered considering for this its two incident edges: e_{i-1} and e_i. Any edge cover of $\mathcal{CE}(G_{i-1})$ containing the edge e_{i-1} (α_{i-1} cases) has already covered v_{i-1} and then the ocurrence of e_i is optional. But for the edge covers where e_{i-1} does not appear (β_{i-1} cases) the edge e_i must appear in order to cover to v_{i-1}. Then, the number of edge covers where e_i appears is $\alpha_{i-1} + \beta_{i-1}$ and just in α_{i-1} edge covers the edge e_i does not appear. Thus, we obtain the new pair (α_i, β_i) associated with the edge e_i, by applying the Fibonacci recurrence relation.

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

When the search arrives to the last edge e_n of the path, we have obtained the pair $(\alpha_{n-1}, \beta_{n-1})$ and since e_n is a fixed edge, then it has to appear in all edge covers of P_n, that means, $\alpha_n = \alpha_{n-1} + \beta_{n-1}$ and since e_n has not chance of not appearing in any edge cover of P_n then $\beta_n = 0$. We call *recurrence for processing fixed edges* to the recurrence:

$$\alpha_i = \alpha_{i-1} + \beta_{i-1}; \quad \beta_i = 0 \tag{2}$$

In this way, the pair associated with the last edge on a path is: $(\alpha_n, \beta_n) = (\alpha_{n-1} + \beta_{n-1}, 0)$. The series $(\alpha_i, \beta_i), i = 1, ..., n$ built based on the recurrences (1) and (2) allows to compute $NE(P_n)$ in linear time over the number of edges in P_n in accordance with the traversing of the path in depth-first search.

We call a *computing thread* or just a *thread* to a series of pairs $(\alpha_i, \beta_i), i = 1, ..., n$ used for computing the number of edge covers on a trajectory of n distinct and adjacent edges. The computation of each pair $(\alpha_i, \beta_i), i = 1, ..., n$ is done in accordance with the type of edge e_i on the trajectory. Notice that a trajectory could be a subgraph of a more complex graph.

In a path P_n, all nodes have degree 2 or 1. But, when the current edge e_i as well as the previous one e_{i-1} considered in a computing thread are incident to a node v_i which has been already covered or for which $\delta(v_i) > 2$, then the edge e_i can be taken into account or not since v_i has been covered previously, and then the pair (α_i, β_i) is computed from $(\alpha_{i-1}, \beta_{i-1})$ by the recurrence relation:

$$\alpha_i = \alpha_{i-1} + \beta_{i-1}; \quad \beta_i = \alpha_{i-1} + \beta_{i-1} \tag{3}$$

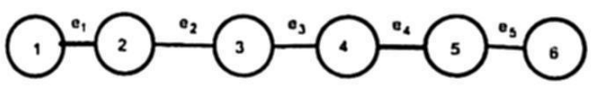
In the following examples, we denote with $\rightarrow$ the application of recurrence (1), with $\xrightarrow{o}$ the application of recurrence (3) and with $\mapsto$ the processing of fixed edges - recurrence (2).

The sequence: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...$, in which each number is the sum of the preceding two, is denoted as the Fibonacci series. 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 from above and from below by $\phi^{i-2} \geq F_i \geq \phi^{i-1}, i \geq 1$, where $\phi = \frac{1}{2} \cdot (1 + \sqrt{(5)})$ is known as the 'golden ratio'.

Example 1 *Let us consider the path P_5 (see figure 2). The computing thread for P_5 is: $(\alpha_i, \beta_i), i = 1, \ldots, 5 : (1,0) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (3,2) \mapsto (5,0)$. Then $NE(P_5) = 5 + 0 = 5$.*

Theorem 1 *The number of edge cover sets of a path of n edges, is:*
$F_n = \texttt{ClosestInteger}\left[\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n\right]$.

Proof. The thread $(\alpha_i, \beta_i), i = 1, \ldots, n$ used for computing $NE(P_n)$, coincides with the Fibonacci numbers: $(F_1, F_0) \rightarrow (F_2, F_1) \rightarrow (F_3, F_2) \rightarrow (F_4, F_3) \ldots \rightarrow (F_{n-1}, F_{n-2}) \mapsto (F_n, 0)$. Then, we infer that $(\alpha_i, \beta_i) = (F_i, F_{i-1})$ for $i = 1, \ldots, n-1$ and $\alpha_n = F_n, \beta_n = 0$. And then $NE(P_n) = \alpha_n + \beta_n = F_n$.



Edges : $e_1 \quad e_2 \quad e_3 \quad e_4 \quad e_5$

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

Fig. 2. Counting edge covers on a path

Case B: Counting Edge Covers on Trees

Let $T = (V, E)$ be a rooted tree. We distinguish each edge on the tree as follows: *root-edges*, which are the edges with one endpoint in the root node; *leaf-edges*, which are the edges with one endpoint in a leaf node of T. Given any intermediate node v of T, 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 traversing by T in post-oder and associating a pair (α_e, β_e) with each edge e of T, except for the leaf edges.

Algorithm #Edge_Covers_in_trees(T)
Input: A rooted tree T with root vertex v_r
Output: $NE(T)$
Procedure:

1. We reduce the input tree T to other tree T' by cutting all leaf nodes and leaf-edges from T, and by labeling as covered nodes all father nodes of the original leaf nodes of T.
2. Traversing by T' in post-order, a pair (α_e, β_e) is associated with each edge e in T'. Each pair is computed in the following way:
   (a) $(\alpha_e, \beta_e) = (1, 1)$ if e is a leaf-edge of T'.
   (b) when an internal node v is visited and it has a set of child-edges, e.g. $u_1, u_2, ..., u_k$ are the child-edges of v, as we have already visited all child-edge of v then each pair $(\alpha_{u_j}, \beta_{u_j})$, $j = 1, \ldots, k$ has been computed and associated to the child-edges. The set of child-edges of v is considered as just one child-edge e_u and its associated pair (α_u, β_u) is computed, as:

$$\alpha_u = \prod_{j=1}^{k}(\alpha_{u_j} + \beta_{u_j}) - \prod_{j=1}^{k}\beta_{u_j} \; ; \quad \beta_u = \prod_{j=1}^{k}\beta_{u_j} \tag{4}$$

   where α_u carries the number of different combinations of the child-edges of v for covering v, while β_u gives the number of combinations among the child-edges of v which do not cover to v. The case in which v has just one child-edge is consider in this case, with $\alpha_u = \alpha_{u_1}$ and $\beta_u = \beta_{u_1}$. The pair associated to the father-edge e_v of v is computed as follows:

$$(\alpha_v, \beta_v) = \begin{cases} (\alpha_u + \beta_u, \alpha_u) & \text{if } v \text{ is a free node or,} \\ (\alpha_u + \beta_u, \alpha_u + \beta_u) & \text{if } v \text{ is a cover node} \end{cases}$$

   This step (2) is iterated until it computes the pairs (α_e, β_e) for all edge e of T' and it stops when it arrives to the root node v_r.
3. Let $(\alpha_{u_r}, \beta_{u_r})$ be the pair associated with the root-edge of v_r. $NE(T)$ is computed according of the status of v_r; if v_r is a covered node then $NE(T) = \alpha_{u_r} + \beta_{u_r}$, otherwise $NE(T) = \alpha_{u_r}$.

This procedure returns $NE(T)$ in time $O(n+m)$ which is the necessary time for traversing T in post-order. Notice that the post-order search allows to give an evaluating order for each edge of T and at the same time, to compute the number of edge covers. We denote with $\succ$ the application of recurrence (4).

Example 2 *Let T be an input tree with root vertex v_r (see fig. 3a). T' is the reduced tree from T where its covered nodes are marked by a black point inside of the nodes (see fig. 3b). T' is traversing in post-order and each pair (α_e, β_e) is associated with each edge e of the tree. 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 by applying recurrence (4), the resulting pair (α_r, β_r) was computed as: $\alpha_r = (1+1)*(4+3)*(6+3) - 1*3*3 = 117$ and $\beta_r = 1*3*3 = 9$. Since v_r is the root node and it is free, then $NE(T) = \alpha_r = 117$.*

Case C: Counting Edge Covers on Simple Cycles

Let $C_n = (V, E)$ be a simple cycle with n edges. Let us order the nodes and edges of C_n, as: $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_n\}, e_i = \{v_i, v_{i+1}\}, i = 1, \ldots, n-1$, $e_n = \{v_n, v_1\}$.

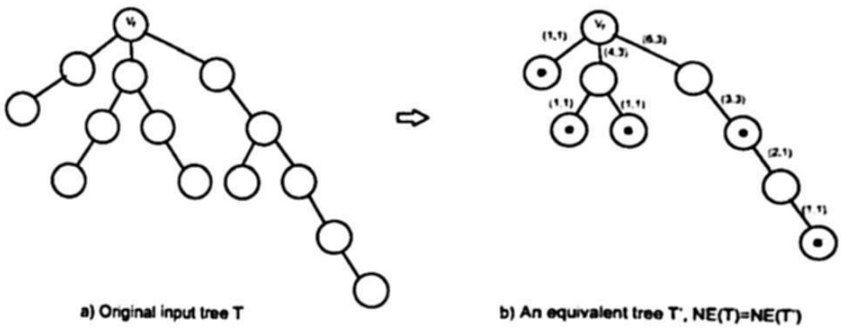


Fig. 3. Computing the number of edge covers for a tree

A computing thread L_p: $(\alpha_i, \beta_i), i = 1, \ldots, n$ is used for counting the edge covers of the path P_n contained in C_n. A depth-first search starts in the edge e_1 and the pair $(\alpha_1, \beta_1) = (1, 1)$ is associated with it, since e_1 is not a fixed edge.

Since all nodes in C_n have degree two, when a new edge is visited during the depth-first search the Fibonacci recurrence (1) is applied. Then, after n applications of recurrence (1), the pair $(\alpha_n, \beta_n) = (F_{n+1}, F_n)$, F_i being the i-th Fibonacci number, is obtained. Let $\mathcal{CE}(NC_n) = \{\mathcal{E} \subseteq E : \mathcal{E}$ is one of the edge sets counted through the thread $Lp\}$, then $NC_n = \mid \mathcal{CE}(NC_n) \mid = \alpha_n + \beta_n = F_{n+2}$.

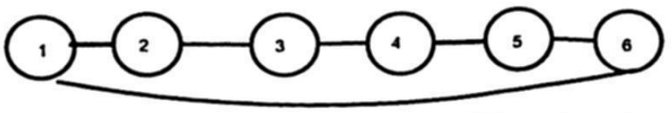
There are some edge sets $\mathcal{E} \in \mathcal{CE}(NC_n)$ where the edges e_1 and e_n do not appear, since the computation of the thread Lp allow these cases, although when the values $\beta_1 = 1$ and $\beta_n > 0$ represent not edge covers for C_n because they do not cover the node v_1. Then, in order to count only the edge covers of C_n, we have to substract from NC_n the edge sets that do not cover v_1.

Let $Y = \{\mathcal{E} \in \mathcal{CE}(NC_n) : e_1 \notin \mathcal{E} \wedge e_n \notin \mathcal{E}\}$ be the edge sets which cover all nodes of C_n except v_1, then $NE(C_n) = NC_n - |Y|$. In order to compute $|Y|$ a new computing thread $(\alpha'_i, \beta'_i), i = 1, ..., n$, denoted by C'_n, is built. C'_n starts with the pair $(\alpha'_1, \beta'_1) = (0, 1)$ since in this way we consider the edge sets where e_1 does not appear. After n applications of recurrence (1), we obtain as last pair of C'_n, $(\alpha'_n, \beta'_n) = (F_{n-1}, F_{n-2})$. For considering only the edge sets where neither e_1 nor e_n appear, (α'_n, β'_n) is taken as $(0, \beta'_n) = (0, F_{n-2})$, and $\mid Y \mid = F_{n-2}$. Then, $NE(C_n) = NC_n - |Y| = F_{n+2} - F_{n-2}$ and we deduce the following theorem.

Theorem 2 *The number of edge cover sets of a simple cycle C_n with n edges, expressed in terms of Fibonacci numbers, is: $NE(C_n) = F_{n+2} - F_{n-2}$.*

In the following examples, we denote with ' $\curvearrowright$ ' the binary operation between pairs: (α_n, β_n) and (α'_n, β'_n) - the final pairs of the two computing threads of a cycle - whose result is the pair: $(\alpha_n, \beta_n - \beta'_n)$ and which has to be assocciated with the last edge e_n of a cycle C_n. Notice that the computation of $NE(C_n)$ has a time complexity of $O(n)$ since we compute the two threads: Lp and C'_n in parallel while the depth-first search is applied.

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



$$
\begin{array}{rl}
 & (\alpha_1,\beta_1) \rightarrow (\alpha_2,\beta_2) \rightarrow (\alpha_3,\beta_3) \rightarrow (\alpha_4,\beta_4) \rightarrow (\alpha_5,\beta_5) \rightarrow (\alpha_6,\beta_6) \\
Lp: & (1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3) \rightarrow (8,5) \rightarrow (13,8) \\
C'_6: & (0,1) \rightarrow (1,0) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3) \\
 & \Rightarrow (13,8) \frown (5,3) = (13,5)
\end{array}
$$

Fig. 4. Obtaining the number of edges covers of a cycle

4 Counting Edge Cover Sets for General Graphs

Let $G = (V, E)$ be a connected graph with $|V| = n$, $|E| = m$ and such that $\Delta(G) \geq 2$. Choose a node $v_r \in V$ (e.g. the node with minimum degree in V) for starting a depth-first search over G in order to build a spanning tree T_G where v_r is the root node.

The edges in T_G are called *tree edges*. While the edges in $(E - E(T_G))$ are called *back edges*. Given a back edge e, the union of the path in T_G between the endpoints of e and the same edge e forms a simple cycle, such cycle is called a *fundamental cycle* of G with respect to T_G. Then, each back edge embraces the maximum path contained in a fundamental cycle. Let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be the set of fundamental cycles found during the depth first search of G. Given any pair of fundamental cycles C_i and C_j from $\mathcal{C}$, if C_i and C_j share edges, we call them *intersecting* cycles; otherwise, they are called *independent* cycles.

We call *critical point* of the graph G to each incident node v_p to a back edge. Given a critical point v_p of G the set of its incident edges $E_p \subset E$ is called a *critical edge set*. Before presenting our most general counting algorithm, let us show the method used for processing critical edge sets of a graph.

Case D: Processing Critical edge sets of a Graph

A main computing thread $Lp : (\alpha_i, \beta_i)$, $i = 1, \ldots, m$, is used for counting the edge cover sets of an input graph G. When an edge $e_i \in G$ is visited for the first time during the depth-first search (called the current edge), its associated pair (α_i, β_i) is computed according to the corresponding recurrence with the type of edge that e_i is in G. However, the series L_p counts all subsets of a critical edge set S of G including the case when all edges of S do not appear. Although for this latter case (for the empty subset of S) the critical point associated with S is not covered.

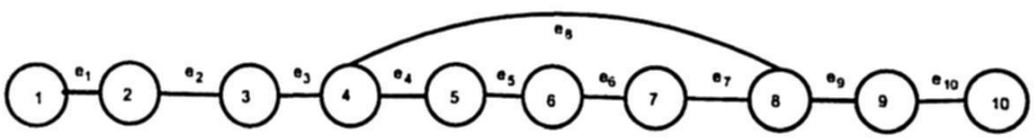
Then, when a critical point v_p is visited for fist time, we have to open a new computing thread in order to count the number of subsets considered by L_p and where all edges of E_p do not appear, and such number has to be subtracted from the current value of L_p. Notice that this case is a generalization of the processing of a back edge for simple cycles. The computing thread L_p is always active until

the counting process finishes, while the subordinated threads are active until all edges of their corresponding critical edge set have been visited.

If a new critical point u_p is visited before visiting all edges of a previous critical edge set E_p, then some new auxiliary threads are created in order to count the number of subsets counted by the current computing threads and for which all edges of the new critical edge set E_u do not appear. In fact, for each active thread l_i (called a main thread of E_u) one auxiliary thread subordinated to l_i is created with initial pair $(0, \beta_u^{l_i})$ being $\beta_u^{l_i}$ equals to β_u in its respective main thread l_i.

When all edges of a critical edge set E_p had already visited, then its associated critical point v_p has been processed and the binary operator $\curvearrowleft$ is applied between the main and its corresponding subordinated thread. After applying $\curvearrowleft$, the subordinated threads created for proccesing E_p are closed and they stop to be active, keeping only the main threads of E_p.

We illustrate in figure 5, how to process critical edge sets. The initial graph G is formed by a cycle union a path on each endpoint of the back edge. The method used for processing the two critical edge sets of G is representative of the way of processing critical edge sets. For this graph, e_8 is the unique back edge. The two critical points in G are v_4 and v_8 which are the endpoints of the back edge. The two critical edge sets are: $E_1 = \{e_3, e_4, e_8\}$ and $E_2 = \{e_7, e_8, e_9\}$.



$$
\begin{array}{lllllll}
\textit{Edges}: & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\
L_p: & (1,0) \rightarrow (1,1) & \rightarrow (2,1) & \xrightarrow{o} & (3,3) \rightarrow (6,3) & \rightarrow (9,6) & \\
L_{E_1}: & & & (0,1) \xrightarrow{o} & (1,1) & & \\
& & & \Rightarrow & (0,1) \rightarrow (1,0) & \rightarrow (1,1) &
\end{array}
$$

$$
\begin{array}{llllll}
\textit{Edges}: & e_7 & e_8 & e_9 & e_{10} & \\
L_p: & (15,9) \xrightarrow{o} (24,24) & \curvearrowleft (0,3) = (24,21) \xrightarrow{o} (45,45) & \curvearrowleft (0,8) = (45,37) & \mapsto (82,0) & \\
L_{E_1}: & (2,1) \xrightarrow{o} (3,3) & \Rightarrow (0,3) & & & \\
e_7 \rightarrow L_p: & (0,9) \xrightarrow{o} (9,9) & & & & \\
& \Rightarrow (0,9) & \curvearrowleft (0,1) = (0,8) \xrightarrow{o} (8,8) & \Rightarrow & (0,8) & \\
e_7 \rightarrow L_{E_1}: & (0,1) \xrightarrow{o} (1,1) & \Rightarrow (0,1) & & &
\end{array}
$$

Fig. 5. Computing NE(G) on a cycle combined with a path

The main thread L_p starts with $(\alpha_1, \beta_1) = (1,0)$ since e_1 is a fixed edge. The recurrence (1) is applied for computing (α_2, β_2) and (α_3, β_3) since v_2 and v_3 have degree 2. Since e_3 is member of the critical edge set E_1, we have to count the edge sets not containing E_1 and which are being considered by L_p. Then, the

auxiliary thread L_{E_1} is created and it is subordinated to L_p. The initial pair for L_{E_1} is $(\alpha'_3, \beta'_3) = (0, \beta_3) = (0, 1)$ since $\beta_3 = 1$ in L_p.

When e_4 is visited, recurrence (3) is applied over each active threads: L_p and L_{E_1}, since $\delta(v_4) > 2$. Although, as we want to count in L_{E_1} only the edge sets where e_3, e_4 (and in advance e_8) do not appear, then (α'_4, β'_4) is changed to $(0, \beta'_4)$, and in this case $(\alpha'_4, \beta'_4) = (0, 1)$.

Next, recurrence (1) is applied until arriving to edge e_7 since the nodes v_5, v_6 and v_7 have degree two. As e_7 is member of the critical edge set E_2, we must count the edge sets not containing E_2 and which are considered by the active threads. Then, two new auxiliary threads are created. The thread denoted by $e_7 \rightarrow L_p$ which is subordinated to L_p and $e_7 \rightarrow L_{E_1}$ which is subordinated to L_{E_1}. Their corresponding starting pairs are: $(0, 9)$ and $(0,1)$ since 9 and 1 are the edge sets counted by L_p and L_{E_1} respectively, where e_7 does not appear.

When e_8 is visited, the recurrence (3) is applied over each active computing thread since $\delta(v_8) > 2$. Since all edges of E_1 have been visited, meaning that the first critical point of G has been processed, the operator $\curvearrowleft$ is applied between L_p and L_{E_1}, as well as between $e_7 \rightarrow L_p$ with $e_7 \rightarrow L_{E_1}$. After to apply the operator $\curvearrowleft$, the threads L_{E_1} and $e_7 \rightarrow L_{E_1}$ are closed and they stop to be active.

As the thread $e_7 \rightarrow L_p$ has been used for counting the sets not containing the edges of E_2, then its corresponding pair (α'_8, β'_8) is changed to $(0, \beta'_8)$. When e_9 is visited, the recurrence (3) as well as the operator $\curvearrowleft$ are applied between the two active threads since all edges of E_2 have already been visited, remaining only the main thread L_p. Finally, when e_{10} is visited, the recurrence (2) is applied and the last pair is $(\alpha_{10}, \beta_{10}) = (82, 0)$. And then, $NE(G) = \alpha_{10} + \beta_{10} = 82$.

One of the main techniques for counting objects on a graph has been the Markov chain Monte Carlo method [1, 2, 5, 3]. Although it is likely that this approximation technique provides efficient algorithms only for graphs with bounded maximum degree. For example, the Markov chain Monte Carlo procedures for counting the number of independent sets of a graph is likely to fail for graphs of maximum degree six or higher [5]. In our case, we are presenting deterministic and exact procedures for counting the number of edge covers of a graph based on the topological structure of the graph and without consideration of its maximum degree. Note that the basic procedures (A), (B) and (C) allow us to compute the number of edge cover sets for any graph with maximum degree 2.

5 Estimating the Relevance of Communication Lines

Complex networks, modeled as large graphs, received much attention during these last years. However, topological information on these networks is only available through intricate measurement procedures. Until recently, most studies assumed that these procedures eventually lead to samples large enough to be representative of the whole, at least concerning some key properties. An important application of counting edge covers is for estimating the degree of reliability in communication networks [10].

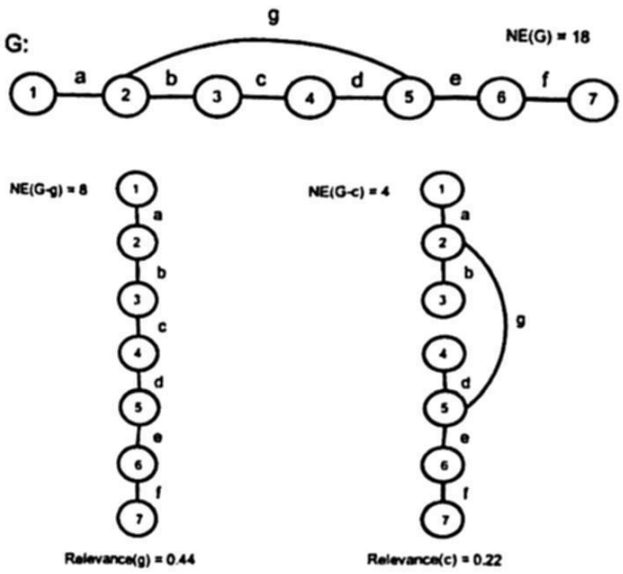


Fig. 6. Estimating the relevance of lines g and c

Table 1. Estimating the relevance of the lines of G

Without line	Edge covers	Relevance
a	15	0.17
b	7	0.61
c	4	0.77
d	6	0.66
e	8	0.55
f	10	0.44
g	8	0.55

For example, if we assume that the communication lines (edges) in a network G has the same 'failure probability' and those failures are independent of one another, we can measure different classes of reliability of the network, given that an edge $c \in G$ fails, according to what component of the network is considered. A way to estimate the 'relevance' of a line c in the network G is by applying the conditional probability $P_{c/G}$ which can be approximated by the fraction of the number of edge covers which are substracted when the edge c is removed (fails), that is, $P_{c/G} = 1 - \frac{NE(G-c)}{NE(G)}$. Thus, $P_{c/G}$ gives the strategic value of an edge c in a network G by estimating the relevance of the line. As c is any edge of G then $P_{c/G}$ could be used for estimating the relevance for any edge of G.

Then, the measure $P_{c/G}$ could be used for estimating the strategic value of any edge c of a graph G with respect to the other lines in G. In such a way that for greater values of $P_{c/G}$ means that the line c is most relevant for maintaining the connectivity of G, in case of failures, with respect to the other lines in G.

6 Conclusions

We determine different recurrence relations for counting, in incremental way, the number of edge cover sets of a graph according with each node and edge of the graph is being visited for the first time during a depth-first search. If the input graph has simple topologies, like: paths, trees, simple cycles or combination of the previous topologies, we can compute the number of edge covers in linear time in the size of the graph.

The class of graphs for which our novel efficient algorithms compute their number of edge cover sets determines a class of polynomial instances for counting the number of edge covers, and such class is a superclass of graphs of degree two without restrictions on the degree of the graphs, but rather, it depends on the topological structure of the graphs.

To know how to compute the number of edge covers is helpful for estimating the reliability of a communication network. For example, we have shown how to estimate the 'relevance' of any line c of the network based on the proportion of the number of edge covers where c does not appear with respect to the total number of edge covers in the network.

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. Bubley R., Dyer M., Greenhill C., Jerrum M., On approximately counting colourings of small degree graphs, *SIAM Jour. on Computing*, 29, (1999), pp. 387-400.
3. Bubley R., Randomized Algorithms: Approximation, Generation, and Counting, Distinguished dissertations Springer, 2001.
4. 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.
5. Dyer M., Greenhill C., Some #P-completeness Proofs for Colourings and Independent Sets, Research Report Series, University of Leeds, 1997.
6. Garey M., Johnson D., Computers and Intractability a Guide to the Theory of NP-Completeness, W.H. Freeman and Co., 1979.
7. Greenhill Catherine, The complexity of counting colourings and independent sets in sparse graphs and hypergraphs", *Computational Complexity*, 9(1): 52-72, 2000.
8. Roth D., On the hardness of approximate reasoning, *Artificial Intelligence 82*, (1996), pp. 273-302.
9. Tarjan R., Depth-First Search and Linear Graph Algorithms, *SIAM Journal on Computing*, Vol. 1, pp.146-160, 1972.
10. Vadhan Salil P., The Complexity of Counting in Sparse, Regular, and Planar Graphs, *SIAM Journal on Computing*, Vol. 31, No.2, pp. 398-427, 2001.