Generating edge covers of path graphs

J. Raymundo Marcial-Romero, J. A. Hernández, Vianney Muñoz-Jiménez and Héctor A. Montes-Venegas
Facultad de Ingeniería, Universidad Autónoma del Estado de México, UAEM, Toluca, México

Abstract— *It is known that the edge cover problem is #P complete. Even for path graphs with m edges it has been shown that the set of edge covers is equal to $fibonnaci(m)$. As a consequence, generating the set of edge covers of a given path graph is an exponential combinatorial problem. In this paper we show that the set of edge covers of a given path graph can be generated by what we call a set of kernel strings. Even more, we show that both the set of kernel strings is bounded by a quadratic polynomial and also there is a quadratic polynomial algorithm which generates kernel strings. As a consequence, a particular edge cover can be recovered from a kernel string in polynomial time.*

Keywords: Edge Covers, Graph Theory, Algorithms, Binary Patterns

1. Introduction

An edge cover of an undirected graph is a subset of its edges such that every vertex of the graph is incident to at least one edge of the subset. It is well known that the *edge cover problem* is #P complete. So, heuristic and exact methods have been proposed to count the number of edge covers for classes of graphs. For example, Bezáková et. al. [1] have shown that a Glauber dynamics Markov chain for edge covers mixes rapidly for graphs with degree at most three. De Ita et. al. [2] have shown that the number of edge covers for path graphs with m edges corresponds to $fibonacci(m)$. Although there is a polynomial time algorithm which computes the number of edge covers of a given path graph, generating the set of edge covers of a given path graph is an exponential combinatorial problem. For example a path graph with 60 edges has 1,548,008,755,920 edge covers. All of them are unthinkable useful for any real application. Even more, assuming that we can encode each edge cover on a byte, the storage space needed will be approximately 1TB for a 60 edges path graph. Instead of generating the whole set of edge covers, in this paper we show how to generate what we call kernel edge covers. Kernel edge covers means that any other edge cover can be generated from them. We show that the number of kernel edge covers is quadratic with respect to the number of edges in the graph. Additionally, we present an algorithm which generates the set of kernel edge covers also in quadratic time.

The set of kernel edge covers for path graphs is generated using an efficient algorithm for generating ascending compositions of an integer n in m parts based in the diagram structure proposed by [3]. The technique uses a binary pattern to map integer partitions into binary strings to represent edge covers.

The paper is organized as follow. Section 2 presents the preliminaries. Section 3 deals with generating partitions of an integer n in m parts, Section 4 presents the generation of edge covers of path graphs. Section 5 concludes the paper and future work is presented.

2. Preliminaries

An undirected graph G (i.e. finite, loopless and with no parallel edges) is defined as a tuple (V, E), where V is the set of *vertices* (or *nodes*) and E the set of *edges*. A vertex and an incident edge are said to *cover* each other.

A path graph $P = (V, E)$ consists of a set of nodes $V = \{a_1, a_2, \ldots a_n\}$ and edges built as $e_i = a_{i-1}a_i$, $1 \leq i \leq n$, i.e a path graph has $m = n - 1$ edges.

The neighborhood of a vertex $v \in V$ is the set $N(v) = \{w \in V : vw \in E\}$ and its degree, denoted as $d_G(v)$, is the number of neighbors that v has. The cardinality of a set A will be denoted as $|A|$. Given a graph $G = (V, E)$, $S = (V', E')$ is called a subgraph of G if $V' \subseteq V$ and $E' \subseteq E$. $G - v$ denotes the subgraph obtained from G by deleting v and all incidents edges to v whereas $G \backslash e$ is the subgraph obtained by simply deletion of the edge e.

An $edge\ cover$ for a graph $G = (V, E)$ is a subset of edges $E' \subseteq E$ that covers all nodes of G, that is, for each $u \in V$ there is a $v \in V$ such that $e = uv \in E'$. Let $\mathcal{E}_G = \{E' \subseteq E : E'$ is an edge cover of $G\}$ be the set of $edge\ covers$ for G, and $|\mathcal{E}_G|$ be the number of $edge\ covers$ of G.

3. Partitions of Integers

In this section we introduce partitions of integers and give and efficient algorithm to compute them. In the following section, the use of partition of integers to generate edge covers will be discussed.

Let n be a positive integer. A composition of n is a way of writing n as the sum of positive integers denoted as $n = y_1 + y_2 + ... + y_k$. If the order of integers y_j is irrelevant, this representation is an $integer\ partition$. When $y_1 \leq y_2 \leq ... \leq y_k$ we have an ascending composition. Algorithms for enumerating all the partitions of an integer or only the partitions with a restriction have been extensively studied [4], [5].

A data structure called *partition diagram* for storing all the partitions of an integer is proposed in [3]. In Merca [6], [7] improvements are proposed which, to date, are the most adequate data structures for generating integer partitions. We use the data structure proposed by Merca to present an efficient algorithm for generating ascending compositions of an integer n in m parts (See Algorithm 1). This algorithm is the foundation to generate edge covers of a path graph.

3.1 Partition Diagram

The partition diagram of an integer n is a directed acyclic graph. Fig. 1 shows a partition diagram of integer $i = 6$. A node in a partition diagram is denoted by (y, Y), where y is an element of a partition and Y is a number to be divided into parts that are not smaller than y. A node (y, Y) that has no predecessor is called a $anchored\ node$ (*root node*) in a partition diagram. A node (y, Y) which has no succesor and $Y = 0$ is called a $terminal\ node$. A node (y, Y) with $Y > 0$ and $y \leq Y$ is called an $internal\ node$. For example, in Fig. 1 the node $(1, 6)$ is an $anchored\ node$ and also $internal\ node$, whereas node $(5, 0)$ is a $terminal\ node$ ($leaf\ node$). Fig. 1 shows a path pointing from $(2, 4)$ to $(2, 2)$ and $(4, 0)$, it means that the nodes $(2, 2)$ and $(4, 0)$ are the succesor of $(2, 4)$.

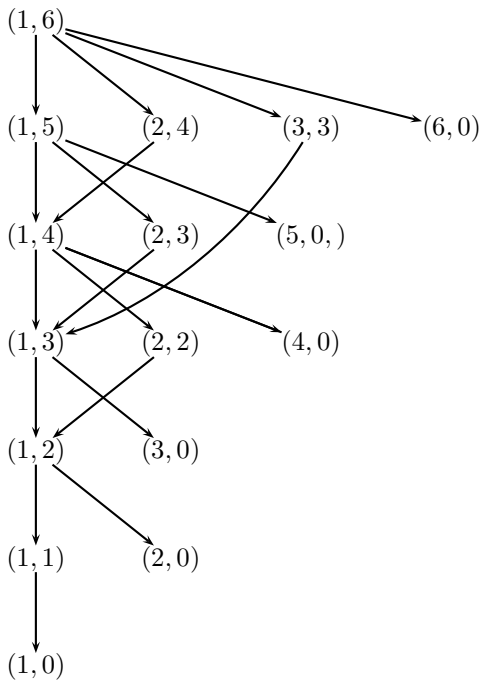

Fig. 1: Partition diagram needed to generate the integer partitions of the number 6. The diagram represents a directed acyclic graph.

Given a partition diagram of an integer n, a path from an anchored node to a terminal node defines a unique partition of n. The partition is formed during a path traversal by recovering all the first parts of the tuples excluding the anchored node and nodes of the form $(1, r)$ if its predecessor is of the form (y, r), where $y > 1$. For example, the path $(1, 6)$ $(1, 5)$ $(1, 4)$ $(2, 2)$ $(1, 2)$ $(2, 0)$ defines partition $\langle 1, 1, 2, 2\rangle$ since the nodes $(1, 6)$ and $(1, 2)$ are excluded. Traversing all the paths, the partitions of 6 are $\langle 1, 1, 1, 1, 1, 1\rangle$, $\langle 1, 1, 1, 1, 2\rangle$, $\langle 1, 1, 1, 3\rangle$, $\langle 1, 1, 2, 2\rangle$, $\langle 1, 1, 4\rangle$, $\langle 1, 2, 3\rangle$, $\langle 1, 5\rangle$, $\langle 2, 2, 2\rangle$, $\langle 2, 4\rangle$, $\langle 3, 3\rangle$, and $\langle 6\rangle$.

Although the partition diagram shown in Fig. 1 is created for integer $i = 6$, it also consists of all the partition diagrams of any integer smaller than 6. The partition diagram of any integer r smaller than n is anchored at node $(1, r)$. For example, the node $(1, 5)$ is the anchored node of the partition diagram of 5. In the next subsection we presented an efficient algorithm to generate the ascending composition of an integer n in m parts.

3.2 Algorithm for generating ascending composition of an integer n in m parts

To obtain all the ascending composition of integer n in m parts, we can generate all paths from the root node to the leaf nodes whose deep is $m + 1$ using the partition diagram. To achieve this, we present a variant of Merca algorithm for traversing all the paths with the constraint of m parts. We represent each level of the partition diagram as a dynamic vector, i.e. $diagram[0] = \{(1, 6)\}$, $diagram[1] = \{(1, 5), (2, 4), (3, 3), (6, 0)\}$ and so on. Particular elements of the diagram can be recovered as $diagram[row][column]$ where row represent the level of the diagram and $column$ the position in the level.

The required variables should be initialized as follows. Variables $row \leftarrow 1$ and $colum \leftarrow 0$, these values represent their position in the diagram. The variable $part \leftarrow 0$, is the number of parts in which n has been already divided. Finally, $partition \leftarrow \{\}$ has the elements of a partition. When a partition is formed, the set $partition$ is stored and modified to generate a new partition.

4. Computing edge covers for path graphs

In this section we show how to generate the set of $edge\ covers$ of a path graph using 3.2. Firstly we present how to encode edge covers for path graphs as binary strings.

4.1 Binary strings to represent edge covers

A binary pattern can be used to represent an $edge\ cover$ of a path graph. Let $b_1 b_2 \dots b_m$ be a binary sequence. If $b_i = 1$ then the edge (a_i, a_{i+1}) belongs to the $edge\ cover$ otherwise (i.e. $b_i = 0$) the edge (a_i, a_{i+1}) does not belong to the $edge\ cover$.

Let G be a path graph with n nodes. A binary sequence $b_1 b_2 \dots b_m$, $m = n - 1$ represents an $edge\ cover$ for G if the following conditions hold:

Algorithm 1 Parts (Generating ascending composition de n in m parts)

Require: $diagram$ (Diagram structure of n)
Require: $m, row, column, part, partition$
1: **for all** $i = 0$; $i \leq length(diagram\,[row] - 1)$; $i = i + 1$ **do**
2: $partition = partition \cup first\ component\ (diagram\,[row]\,[i])$;
3: $part \leftarrow part + 1$;
4: **if** $second\ component\ (diagram\,[row]\,[i])! = 0$ and $part < m$ **then**
5: $Parts$ (n-$second\ component\ (diagram\,[row]\,[i]) + 1$, 0, $part$,m,n)
6: **else**
7: **if** $second\ component\ (diagram\,[row]\,[i] = 0)$ and $part == m$ **then**
8: store $partition$;
9: **end if**
10: **end if**
11: $partition = partition$-$last\ component(partition)$;
12: $part \leftarrow part - 1$;
13: **end for**

1) $\nexists\ b_i, b_{i+1}$ such that $b_i = 0$ and $b_{i+1} = 0$.
2) $b_1 = 1$ and $b_m = 1$.

An $edge\ cover$ representation, does not admit a sequence of consecutive zeros. So, a binary sequence is represented as
$1^{p_0}01^{p_1}\cdots 01^{p_{l-1}}01^{p_l}$ for some $l \in \mathbb{N}$ and $p_0, p_1, \ldots p_l > 0$.

Lemma 1: Let ω be a binary sequence which represents an edge cover of a path graph with n nodes. The maximum number s_n of zeros appearing in ω is given by
$s_n = \frac{n-2}{2}$; if n is even and
$s_n = \frac{n-1}{2}$; if n is odd.

Proof:

It is obvious that, when each $p_i = 1$, $1 \leq i \leq n$, the string ω has the maximum number of zeros. Since the one's appearing at the extrema of ω are fixed, then a string of length $n-2$ is left where half of the symbols are zero if n is even or half of $n-1$ symbols are zero if n is odd. ■

Corollary 1: Let P be a path graph such that the number of nodes of P is n. If ω is a binary string which represents an edge cover for P then

1) if n is even there are $0 \leq s \leq \frac{n-2}{2}$ zeros on ω;
2) if n is odd there are $0 \leq s \leq \frac{n-1}{2}$ zeros on ω.

Example 1: : If a path graph with $n = 10$ nodes is given, then $0 \leq s \leq 4$ zeros are allowed in a binary string to represent an $edge\ cover$. If $s = 0$ then, there is only one binary string $\langle 1111111111\rangle$, if $s = 1$ then there are eight binary strings $\langle 1011111111\rangle$, $\langle 1101111111\rangle$, $\langle 1110111111\rangle$, $\langle 1111011111\rangle$, $\langle 1111101111\rangle$, $\langle 1111110111\rangle$, $\langle 1111111011\rangle$, $\langle 1111111101\rangle$ and so on.

It is easy to see that there is only one string without zeros that represent an $edge\ cover$ of a path graph with n nodes. In fact, it is the string with $n-1$ ones (each one represents an edge). The strings with one zero that represent $edge\ covers$ are also easily counted and generated as the following lemma shows.

Lemma 2: Let P be a path graph with m edges ordered as $a_1, a_2, \ldots, a_m$. There are $m-2$ strings of length m with one zero that represent edge covers of P.

Proof: Let $b_1 b_2 \ldots b_m$ be an arbitrary string which represent the edges of the path graph P. It is obvious that b_1 and b_m should be one since each of them cover the nodes a_1 and a_n respectively. So the string where $b_2 = 0$ and $b_i = 1$, $i = 1, 3, 4, \ldots m$ represents an edge cover of P with one zero. If we shift the value of b_2 to b_3 and assign to b_2 the value one, we have the second string that represents an edge cover of P with one zero. In general if $b_i = 0$, $2 \leq i \leq m-2$ then shifting the value of b_i with b_{i+1} gives a new edge cover of P with one zero. ■

The generation of some strings with more that one zero which represent $edge\ covers$ can be determined via the correspondence with the integer partitions of a number n.

Definition 1: A kernel string is a binary string of the form $01^{p1}01^{p2}\cdots 1^{pl}0$, $l \geq 1$ where $0 < p1 \leq p2 \leq \cdots \leq pl$.

Proposition 1: There are $n-4$ kernel strings with two zeros of length at most $n-2$.

Proof: A kernel string with two zeros is of the form $01^{p1}0$. In fact, if $1 \leq p1 \leq n-4$ the kernel strings $01^{p1}0$ have length at most $n-2$. ■

An ascending integer partition can be used to generate kernel strings.

Lemma 3: Let $p(n,m)$ be the set of ascending integer partitions of n in m parts. If $l_1 + l_2 + \cdots + l_m \in p(n,m)$, then $01^{l_1}01^{l_2}\cdots 1^{l_m}0$ is a kernel string with $m+1$ zeros whose length is $l_1 + l_2 + \cdots + l_m + m + 1$.

Proof: That $01^{l_1}01^{l_2}\cdots 1^{l_m}0$ is a kernel string is easily verified since each $l_i \neq 0$ and the ascending condition means that $l_1 \leq l_2 \leq \cdots \leq l_m$. The number of zeros is also straightforward computed. ■

Lemma 4: There are $\sum_{i=r}^{n-r-3} |p(i,r)|$ kernel strings with $r+1$ zeros for all $n \geq 5$.

Proof: The proof is by induction over n. ■

The following example shows how to generate a kernel string from an ascending integer partition $p(n,m)$.

Example 2: The partitions of four in two part are p(4,2)={2+2, 1+3}. Since the cardinality of the set is two, this means that there are two kernel strings with three zeros and four ones. The kernel string are of the form $01^{p1}01^{p2}0$. Each element of a partition is the value of a p_i. So one kernel string is formed when $p1 = p2 = 2$ and the other kernel string is formed when $p1 = 1$ and $p2 = 3$ which are the kernel strings 0110110 and 0101110, respectively. From lemma 4, if $n = 9$ there are four kernel strings with three zeros because

$$\begin{aligned}\sum_{i=2}^{9-2-3} p(i,2) &= \sum_{i=2}^{4} p(i,2) \\ &= p(2,2)+p(3,2)+(4,2) \\ &= 1+1+2 \\ &= 4\end{aligned}$$

The kernel strings are shown in the following table.

$p(2,2)=1+1$	01010
$p(3,2)=1+2$	010110
$p(4,2)=\{2+2,1+3\}$	0110110, 0101110

It is well known that if $l_1+l_2+\cdots+l_m$ is an integer partition of a number n then a combination of the number, i.e., $l_2+l_1+\cdots+l_m$, represents the same integer partition of n. However, if they are used to generate kernel strings, those strings are different. We can not call both of them kernel strings since, if $l_1 \leq l_2$, it is not the case that $l_1 > l_2$, so the second is not a kernel string. We introduce another definition to include those strings.

Definition 2: A combined string is a binary string of the form $01^{p1}01^{p2}\cdots 1^{pl}0$, $l \geq 1$ where each $pi \neq 0$, $1 \leq i \leq l$.

Now, combined strings can be generated from integer partitions also. However, we do not want to count combinations of ascending integer partitions that are identical, i.e., if $l_1+l_2+\cdots+l_m$ is a partition which represent a combined string and $l_1=l_2$ then, $l_2+l_1+\cdots+l_m$ will represent the same string. We use λ_{l_i} to denote the number of times the value l_i is repeated in the partition.

Let $t=l_1+l_2+\cdots+l_m$ be an ascending integer partition of the number t. It is well known that the total number of non-repeated combinations of the partition t is given by $(\sum_{i=1}^{m}\lambda_{l_i})!/\prod_{i=1}^{m}(\lambda_{l_i})!$. Algorithms which efficiently built these kind of integer partition combinations have long been studied, a survey can be found in Knuth [8]. What is important to point out is that combined strings can de generated from kernel string since combined integer partitions can be generated form integer partitions.

Corollary 2: Each combined string can be generated from a kernel string.

Corollary 3: The set of kernel strings is a subset of the set of combined strings.

Lemma 5: Let P be a path graph with m edges. If w is a combined string such that $|w| \leq m-2$ then $1w1^l$ represents an edge cover of P where $l=m-|w|-1$.

Proof: By definition, a combined string does not have two consecutive zeros. That the length of $1\omega 1^l$ is m is establish by the condition $l=m-|w|-1$. ■

Theorem 1: Let P be a path graph with $m \geq 4$ edges.

1) If m is even then $(\frac{m}{2}-2)(\frac{m}{2}-1)$ kernel strings are needed to generate combined strings that represent edge covers.

$$\begin{array}{lll} p(1,1) & & \\ p(2,1) & p(2,2) & \\ p(3,1) & p(3,2) & p(3,3) \\ p(4,1) & p(4,2) & p(4,3) \\ p(5,1) & p(5,2) & \\ p(6,1) & & \end{array}$$

Table 1: Ascending integer partitions calculated from Algorithm 2 for $n=10$

2) If m is odd then $\frac{(m-3)^2}{4}$ kernel strings are needed to generate combined strings that represent edge covers.

Proof: We first prove the case where m is even. By lemma 1, it suffices to generate kernel strings with s zeros, $2 < s < (n-2)/2$. By proposition 1 there are $m-4$ kernel strings with two zeros. In the same way, there are $n-3$ kernel strings with three zeros. In general, there are $(n/2)+1$ kernel strings with $(n-2)/2$ zeros. Adding the number of string we have $\sum_{i=4,even}^{m-2}(m-i)$ kernel strings. $\sum_{i=4,even}^{m-2}(m-i)=(\frac{m}{2}-2)(\frac{m}{2}-1)$. Similarly, if m is odd, there are $\sum_{i=4,odd}^{m-3}(m-i)$ kernel strings and $\sum_{i=4,odd}^{m-3}(m-i)=\frac{(m-3)^2}{4}$. ■

Algorithm 2 Increasing integer partitions needed to generate kernel strings

Require: Integer $m \geq 4$ which represent the number of edges of P

```
if m odd then
    l = (m-3)/2
    p(l,l)
else
    l = (m-2)/2
end if
for all j = 1; j ≤ l; j++ do
    for all i = j; i < 2l - j; i++ do
        p(i,j)
    end for
end for
```

Let P be a path graph with $m \geq 4$ edges, algorithm 2 calculates the ascending integer partitions of l in r parts needed to generate kernel strings based on the number of edges m. For example, if $n=10$, Table 1 shows which ascending integer partitions are needed. We use a tThe integer partitions are presented in rows and columns to see its correspondence with the previous results. For example, the second component of the elements in row one are 1. Those partitions represent the kernel strings with two zeros needed (lemma 4).

Although we known how to compute the set of combined string from kernel strings, there is a last set of strings which represent $edge\ covers$ of path graphs that are not included in lemma 5. For example, the string 10110101111 represents an $edge\ cover$ of a path graph with eleven edges, the combined

string which generates such a string is 011010 based on a combination of the ascending partition $1 + 2$, i.e. $2 + 1$. However, the string 11011010111 which also represents an $edge\ cover$ of a path graph with eleven edges is not generated from the result of lemma 5. Those strings can be obtained by shiftings to the right combined strings in the whole string. In our example, if the combined string 011010 is shifted one position to the right in the string 10110101111 , the string 11011010111 is generated.

Lemma 6: Let P be a path graph with m edges. If $1w1^l$ represents an edge cover for P where w is a combined string and $l > 1$ then, $l - 1$ different strings which represent edge covers of P can be generated from $1w1^l$.

Proof: Since $l > 1$ then, shifting one place to the right the combined string w generates a new string $11w1^{l-1}$ which also represents an edge cover of P. It is straightforward to notice that the shifting can be done $l - 1$ times. ■

We are now in a position to present the algorithm to generate the strings that represent $edge\ covers$ for a path graph with at least four edges. $Edge\ covers$ for path graphs with less than four edges are straightforward calculated.

Algorithm 3 Generating the edge covers of a path graph P

Require: Integer $m \geq 4$ which represent the number of edges of P

```
Generate diagram structure of m
Include the string without zeros, i.e., 1^m
Include m − 2 edge covers with one zero
Compute kernel strings
Generate combined strings
for all combined string w do
    p = m − |w| − 1
    for all i = 0; i < p; i++ do
        add 11^i w 1^(p−i)
    end for
end for
```

5. Conclusions and future work

A procedure to compute $edge\ covers$ for $path\ graphs$ using an efficient algorithm for generating ascending compositions of an integer n in m parts has been presented. Although the number of edge covers grows exponentially [2], the space and time needed to store either the partition diagram or the set of kernel strings is quadratic. The implication of this result is that, in practical applications, we can efficiently recover subsets of edge covers for a given path graph as kernel strings are the base to generate combined strings. We believe that the procedure presented will be the foundation to generate edge covers for simple graphs, i.e., acyclic and cyclic.

References

[1] I. Bezáková and W. Rummler, "Sampling edge covers in 3-regular graphs," in *Mathematical Foundations of Computer Science 2009*, ser. Lecture Notes in Computer Science, R. Královic and D. Niwiński, Eds. Springer Berlin Heidelberg, 2009, vol. 5734, pp. 137–148.

[2] G. D. Ita, J. R. Marcial-Romero, and H. A. Montes-Venegas, "Estimating the relevance on communication lines based on the number of edge covers," *Electronic Notes in Discrete Mathematics*, vol. 36, no. 0, pp. 247 – 254, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1571065310000338

[3] R.-B. Lin, "Efficient data structure for storing the partitions of integers," *The 22nd Workshop on Combinatorics and Computation Theory*, pp. 349–354, 2005.

[4] D. Stanton and D. White, "Constructive combinatorics," *Springer-Verlang, Berling*, 1986.

[5] C. L. Liu, "Introduction to combinatorial mathematics," *MacGraw-Hill College*, 1986.

[6] M. Merca, "Binary diagrams for storing ascending compositions," *The Computer Journal Advance Access*, 2012.

[7] ——, "Fast algorithm for generating ascending compositions," *Journal of Mathematical Modelling and Algorithms*, vol. 11, pp. 89–104, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10852-011-9168-y

[8] D. E. Knuth, *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Addison Wesley, 2011.