JAM CM Journal of Applied Mathematics and Computational Mechanics 2018, 17(2), 17-27
www.amcm.pcz.pl p-ISSN 2299-9965
DOI: 10.17512/jamcm.2018.2.02 e-ISSN 2353-0588

GRAPH PRESENTATION OF BINARY STRINGS

Qefsere Doko Gjonbalaj, Valdete Rexhëbeqaj Hamiti

Department of Math, Faculty of Electrical and Computer Engineering
University of Prishtina "Hasan Prishtina"
Prishtinë, 10 000, Kosovë
qefsere.gjonbalaj@uni-pr.edu, valdete.rexhebeqaj@uni-pr.edu

Received: 31 January 2018; Accepted: 26 April 2018

Abstract. This article is motivated by the problem of finding a graph, which operates on the principle of the Hasse diagram. In the present article, the Hasse diagram technique (HDT) was applied to relate binary string sets with graph theory. We investigate this relation in detail and propose a new graph, the so-called *2n-Parallel Graph* $(2nPG)$, and an efficient pseudo code that decompresses a given binary string set to its elements. The goal of this pseudo code is its application in determining the number of strings with a certain number of first bits consecutive 1's (0's). This pseudocode is also responsible for the solution of several combinatorial problems on a certain binary string. Furthermore, our results are valid for any string length.

MSC 2010: *05C90, 68R10*
Keywords: *Hasse diagram, binary strings*

1. Introduction

As we know, binary strings can be decoded using a binary tree. To read that string from any prescribed position forward, one starts at the root and moves down from the root of the tree to the external node that holds that character: a 0 bit identifies a left branch in the path, and a 1 bit identifies a right branch. This procedure is continued until the bit string of length n is decoded. But with an increase in the length of the string, the binary tree becomes more complicated [1].

So we look for another way to represent binary strings that will allow us to minimize the number of branches in our drawing. Looking to simplify the problem we have defined a new graph that works by use of the Hasse diagram technique.

2. Basic notions

All graphs considered here are simple, finite, connected and undirected. We follow the basic notions and terminologies of graph theory as in [1].

Definition 2.1. A *graph* $G = \big(V(G), E(G)\big)$ consists of two sets; $V(G) = \{v_1, v_2, \ldots\}$ called *vertex set* of G and $E(G) = \{e_1, e_2, \ldots\}$ called *edge set* of G. More precisely we denote the vertex set of G as $V(G)$ and the edge set of G as $E(G)$. Elements of $V(G)$ and $E(G)$ are called *vertices* and *edges,* respectively. The number of vertices in $V(G)$ is denoted by $|V(G)|$ and the number of edges in $E(G)$ is denoted by $|E(G)|$.

Definition 2.2. A walk in which no vertex is repeated is called a simple path. A path with n vertices is denoted as P_n.

Definition 2.3. A graph G is connected if there is a path between every pair of vertices of G. A graph, which is not connected, is called a disconnected graph.

Definition 2.4. [2] A graph G with $n \geq 2$ vertices is called a path if G has exactly two vertices with degree 1 and exactly $n - 2$ vertices with degree 2.

Definition 2.5. A simple path in a graph G that passes through every vertex exactly once is called a Hamilton path, and a simple circuit in a graph G that passes through every vertex exactly once is called a Hamilton circuit.

Theorem 2.1. [3] Let G be a finite connected graph on n vertices. Then G contains at least $n - 1$ edges, with equality if and only if G is a tree.

Here we review some basic definitions and notions concerning posets (partially ordered set) and the Hasse diagram.

Definition 2.6. [2] A Hasse diagram H is a directed graph representation of a poset P whose vertices represent elements of P and whose directed edges signify the " $\leq$ " relation. That is, an edge $v \to \omega$ means $v \leq \omega$.

In other words, the Hasse diagram H of poset P is defined:

(1) $V(H) = P$

(2) $E(H) = \big\{v \to \omega \big| v, \omega \in P, v \leq \omega, \nexists t \in P \; such \; that \; v \leq t \leq \omega\big\}$

and in this case we say that ω covers v, so that the line segment from v to ω runs upwards P.

If we assume that all edges are pointed “upward”, we do not have to show the directions of the edges.

The objects (vertices) in a Hasse diagram that are not covered by other objects are called “maximal objects.” Objects that do not cover other objects are called “minimal objects.” In some diagrams there are also isolated objects that can be considered as maximal and minimal objects at the same time. A chain (string) is a set of comparable objects; therefore levels can be defined as the longest chain within the diagram. An antichain is a set of mutually incomparable objects, located at one and the same level. The height of the diagram is the longest chain, and the longest antichain is its width [4].

3. 2*n* Parallel Graph (2*nPG*)

Let Σ be the set {0,1}. Then the set of all finite binary strings of length n is written as Σ_n^*. Σ_n^* can be ordered by the prefix relation: for $u, v \in \Sigma_n^*$, u is a prefix of v if either $u = v$ or u is a finite initial substring of v. We write $u \leq v$ if u is a prefix of v and $u \| v$ if neither u nor v is a prefix of the other (some authors write $u \nparallel v$). For each set of binary strings of length n there is $|\Sigma_n^*| = 2^n$.

For brevity, we give the following definition.

Definition 3.1. [5] Let $s \equiv a_1, a_2, \dots, a_n$ be an element of Σ_n^*. Weight of s, denoted by $\omega(s)$, is defined as $\omega(s) = \sum_{i=1}^{n} a_i$.

For example, the string 10011 has weight 3.

Now we are going to define an efficient graph (which operates on the principle of Hasse diagram) for the set of binary strings of length n whose weight (number of 1s) are in the range $c, c+1, c+2, \dots, d$ where $0 \leq c < d \leq n$.

Definition 3.2. 2n *Parallel Graph* $(2nPG)$ is a graph consisting of $2n$ vertexes situated in the proportional way in to parallel line. The vertices in first line will be denoted with u_i where $i = 1,2, \dots, n$ and $u_i = 1$ for $\forall i$, while vertices in second line with v_i where $i = 1,2, \dots, n$ and $v_i = 0$ for $\forall i$. The set of $(2nPG)$ edges in graph will be defined with $E = \{\{u_i, u_{i+1}\}, \{v_i, v_{i+1}\}, \{u_i, v_{i+1}\}, \{v_i, u_{i+1}\}\}$, for $i = 1,2, \dots, n-1$ and all edges are pointed "upward", which means it works by the Hasse diagram technique.

From the definition of graph $(2nPG)$ it follows that the set of vertices V are defined by $V = \{u_1, u_2, \dots, u_n, v_1, v_2, \dots, v_n\} = \{1,1, \dots, 1,0,0, \dots, 0\}$ and $|V| = 2n$.

The problem of enumeration of all n-bit binary strings is equivalent to finding a Hamilton path in the $(2nPG)$ graph. Because of the Hasse diagram technique, the Hamilton path in $(2nPG)$ graph contains only n vertices (not 2n).

Thinking of binary strings as path in the graph, based on Theorem 1.1, the strings of length n are mapped to a path of length $n-1$. So, from this fact we conclude that:

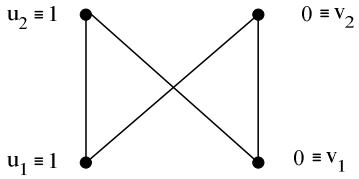
Theorem 3.1. Let Σ_n^* be the set of all finite binary strings of length n, where $|\Sigma_n^*| = 2^n$. Then, the graph $(2nPG)$, which represents the element of set Σ_n^* for $n \geq 2$ has $4(n-1)$ edges, which means that $|E(2nPG)| = 4(n-1)$.

Proof: From the definition 3.2, the set of edges of $(2nPG)$ graph is the set $E = \{\{u_i, u_{i+1}\}, \{v_i, v_{i+1}\}, \{u_i, v_{i+1}\}, \{v_i, u_{i+1}\}\}$, for $i = 1,2, \dots, n-1$. Therefore, from the Theorem 1.1, every subset of set E will have $n-1$ edges and since E has four such subsets then follows that $|E(2nPG)| = 4(n-1)$.

Following, based on the definition 3.2, we will construct a $(2nPG)$ graph and compare it with binary trees. By representing the set of binary strings with both graphs: with $(2nPG)$ and by binary tree, we can see the advantages and simplicity of the first graph to the second graph.

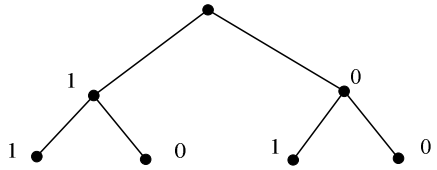
Example 3.1. We are going to present the set of binary strings with the $(2nPG)$ graphs and with the binary trees, for $n = 2,3,4$. Here, for the $(2nPG)$ graph we will use the Hasse diagram technique.

For $n = 2$ we have (Fig. 1a and 1b):



$|V| = 4 = 2 \cdot 2, |E| = 4(n-1) = 4$

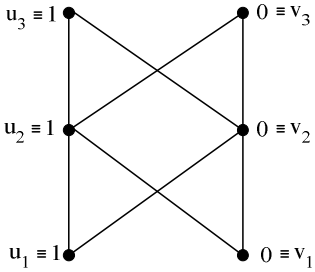
Fig. 1a. $(2nPG)$ graph for binary strings of length $n = 2$



$|V| = 7 = 2^{n+1} - 1, |E| = 6 = 2^{n+1} - 2$

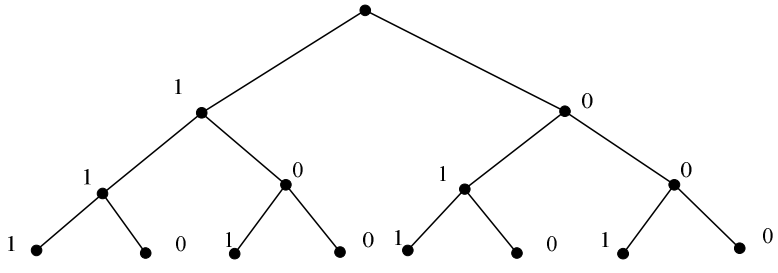
Fig. 1b. Binary tree for binary strings of length $n = 2$

In this case we get $2^n = 2^2 = 4$ binary strings: 11, 10, 01, 00.
For $n = 3$ we have (Fig. 2a and 2b):



$|V| = 6 = 2 \cdot 3, |E| = 4(n-1) = 8$

Fig. 2a. $(2nPG)$ graph for binary strings of length $n = 3$

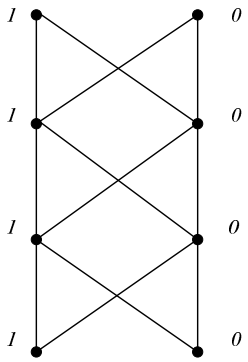


$|V| = 15 = 2^{n+1} - 1, |E| = 14 = 2^{n+1} - 2$

Fig. 2b. Binary tree for binary strings of length $n = 3$

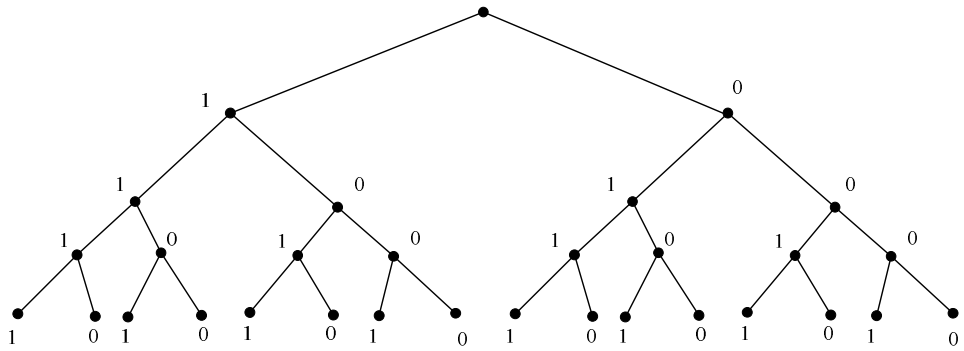
In this case we get $2^n = 2^3 = 8$ binary strings: $111, 110, 101, 100, 000, 001, 010, 001$.

For $n = 4$ we have (Fig. 3a and 3b):



$|V| = 8 = 2 \cdot 4, |E| = 4(n-1) = 12$

Fig. 3a. $(2nPG)$ graph for binary strings of length $n = 4$



$|V| = 31 = 2^{n+1} - 1, |E| = 30 = 2^{n+1} - 2$

Fig. 3b. Binary tree for binary strings of length $n = 4$

In this case we get $2^n = 2^4 = 16$ binary strings: 1111, 1110, 1101, 1100, 1011, 1001, 1010, 1000, 0000, 0001, 0010, 0011, 0100, 0110, 0101, 0111.

We can continue to construct, for each n, a Hamilton path in a $(2nPG)$ graph with the following additional property: edges between levels $i-1$ and i of a $(2nPG)$ graph must appear on the path before edges between levels i and $i+1$.

Corollary 3.1. Let Σ_n^* be the set of all finite binary strings of length n, where $|\Sigma_n^*| = 2^n$. Then, for $n \geq 2$, the number of edges of a $(2nPG)$ graph creates the arithmetic sequence $\{4(n-1)\}$ while the number of edges in a binary tree creates the sequence $\{(2^{n+1} - 2)\}$.

Proof: From the example 3.1, we note that the sequence of number of a $(2nPG)$ graph is: $4, 8, 12, \ldots, a_{n-1} + 4$ or the arithmetic sequence with first element $a_1 = 4$ and difference $d = 4$ (it can be proved with mathematical induction). Also the number of edges of binary trees creates the sequence: $6, 14, 30, \ldots, 2^{n+1} - 2$.

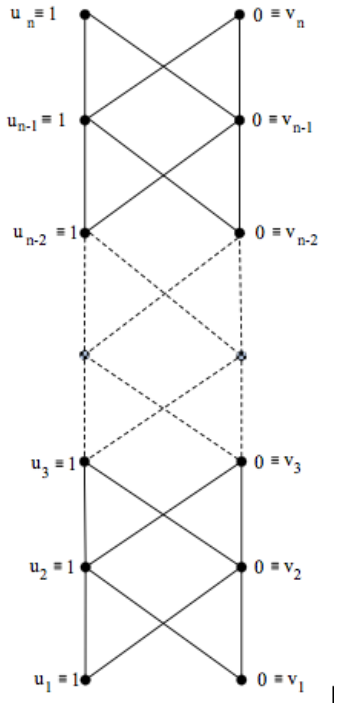


Fig. 4. $(2nPG)$ graph for binary strings of length n

Corollary 3.2. Let Σ_n^* be the set of all finite binary strings of length n, where $|\Sigma_n^*| = 2^n$. Then, $n \geq 2$, the number of vertices of a $(2nPG)$ graph creates the arithmetic sequence $\{2n\}$ while the number of vertices of binary trees creates the sequence $\{(2^{n+1} - 1)\}$.

Proof: From the example 3.1, we note that the sequence of number of vertices of a $(2nPG)$ graph is: $4, 6, 8, \dots, a_{n-1} + 2$ or the arithmetic sequence with first element $a_1 = 4$ and difference $d = 2$ (it can be proved with mathematical induction). Also the number of edges of binary trees creates the sequence: $7, 15, 31, \dots, 2^{n+1} - 1$.

In conclusion, based on two corollaries and our own example we can see that the $(2nPG)$ graph, is much more practical because it has a much smaller number of vertices and edges compared with the binary tree, in which the number of vertices and edges is much larger, and it grows very fast. Also, the advantage of this graph is the fact that it is easier to draw and takes less space.

In following we will describe the pseudo code that produces a $(2nPG)$ graph.

4. Pseudocode of a 2*n* Parallel Graph $(2nPG)$

In this section an efficient pseudocode will be provided that produces a $(2nPG)$ graph by decompressing a given binary string set to its elements and how does it

operates on the principle of Hasse diagram. At the beginning, we give a pseudocode and afterwards, we discuss the complexity of our solution.

The problem of enumeration of all n-bit binary strings is equivalent to finding a Hamilton path with n vertices and $n-1$ edges, in the $(2nPG)$ graph, which means that at the same time we will determine the binary strings and Hamiltonian paths.

In that case, we denote with $S_k^n \in \Sigma_n^*$ all binary strings of length n where the first k bits are consecutive 1's (0's).

Our pseudocode for the $2n$ Parallel Graph is presented as a method called 2*NPG*(n) (cf. Fig. 5) which takes as a parameter the length n for each of the binary strings that are to be generated.

2NPG(n)
INPUT: n is the length of each of the binary strings to generate.
OUTPUT: S is the set of generated binary strings S_k^n such that
- each S_k^n is represented with $[e_1,e_2,e_3,..,e_{k-1},e_k,e_{k+1},..,e_{n-1},e_n]$, and
- k represents the nr. of first bits consecutive 1's in the string to generate.

```
1  let initially S ← {}      // initially is S an empty set of binary strings
2  for k ← n to 1            // for each step below, i.e. distinct nr. of consecutive 1's
3     do 2nPG(n, k)          // where k <= n
4     RETURN {S_k^n}← {[e1,e2,e3,…,e_{k-1},e_k,{e_{k+1},...,e_{n-1},e_n}]}
5  RETURN S ← S U{S_k^n}← {[e1,e2,e3,…,e_{k-1},e_k,e_{k+1},...,e_{n-1},e_n]}
```

Fig. 5. The 2*NPG*(n) pseudocode

We represent with $[e_1,e_2,e_3,..,e_{k-1},e_k,e_{k+1},..,e_{n-1},e_n]$ each of the binary strings S_k^n of length n to be generated as elements of the final set S of all binary strings generated at the output. The S set which is initially an empty set (line 1 in the pseudocode) is progressively extended (line 5) for each possible k as first consecutive 1 bits (line 2) in the strings to generate by recursively invoking the 2*NPG*(n, k) method (line 3 to 4).

2NPG(n, k)
INPUT:
- n is the length of each of the binary strings to generate
- k represents the nr. of first bits consecutive 1's in the strings to generate.

OUTPUT: S is the set of generated binary strings S_k^n such that
- each S_k^n is represented with $[e_1,e_2,e_3,\ldots,e_{k-1},e_k,e_{k+1},...,e_{n-1},e_n]$.

```
1  let initially {S_k^n} ← {}          // initially is {S_k^n} an empty set of binary strings
2  for i ← 1 to k
3  do e_i← u_i                         // assign 1 to e_i since u's are always 1's
4  if k ≠ n then e_{k+1}← v_{k+1} // assign 0 to e_{i+1} since v's are always 0's
5  if k < n–1 then for i ← k+1 to n
6              do 2nPG(n') // where n' = n–k; each S is S_k'^{n'}←[e_{k+1},..,e_{n-1},e_n]
7              RETURN {S_k'^{n'}}← {[e_{k+1},...,e_{n-1},e_n]}
8  RETURN {S_k^n}← {S_k^n} U {[e1,e2,e3,…,e_{k-1},e_k,{e_{k+1},...,e_{n-1},e_n}]}
```

Fig. 6. The 2*NPG*(n, k) pseudocode

It is obvious that in the very first iteration of the outer for loop in the 2*NPG*(*n*) pseudocode above where $k = n$, makes the inner for loop in 2*NPG*(*n*, *k*) (lines 3 to 4) generate a string of length *n* with all 1's, i.e. the string S_n^n :

$$[1,1,1,..,1,1,1] \equiv [u_1,u_2,u_3,...,u_{n-2},u_{n-1},u_n]$$

since lines 5 and 6 to 8 fail due to conditions in line 5 and line 6 evaluating to false.

Next, due to $k = n - 1$ (refer again to outer for loop in 2*NPG*(*n*), this time its second iteration), the inner for loop in 2*NPG*(*n*, *k*) (lines 3 to 4) will generate a string with $n - 1$ 1's, whereas the last bit n of the string will be set to 0 (line 5) since the condition in line 5 evaluates to true, i.e. the following string S_{n-1}^n: will be generated at the output:

$$[1,1,1,..,1,1,0] \equiv [u_1,u_2,u_3,...,u_{n-2},u_{n-1},v_n]$$

Note that lines 6 to 8 fail due to condition in line 6 evaluating to false.

In every other next iteration of the outer for loop in 2*NPG*(*n*) starting from the 3rd iteration, i.e., for $k = n - 2$ to $k = 1$, both conditions (lines 5 and 6) in the inner for loop in 2*NPG*(*n*, *k*) are evaluated to true. As a consequence, another for loop is initiated (line 6) which recursively uses the Hasse diagram techniques but for shorter strings of length n' to generate at the output at each iteration by invoking the 2*nPG*(*n*') method (line 7), resulting into a subset $\{S_{k'}^{n'}\}$ of such strings $[e_{k+1},...,e_{n-1},e_n]$ per iteration.

As an illustration, two next iterations in the 2NPG(n) pseudocode of the outer for loop where $k = n - 2, k = n - 3$, up to its last iteration where $k = 1$ are explained.

1. We create the binary strings where the first *n*-2 bits are 1's, the set of strings denoted as $\{S_{n-2}^n\}$:

   $[1,1,1,...,1,0,?] \equiv [u_1,u_2,u_3,...,u_{n-2},v_{n-1},v_n]$ and

   $[1,1,1,...,1,0,?] \equiv [u_1,u_2,u_3,...,u_{n-2},v_{n-1},u_n]$

   where the last bit

   u_n or v_n

   is determined based on the Hasse diagram technique, i.e. are the output of the 2nPG(n') method (line 7) within the 2nPG(n, k) pseudocode.
2. We create the binary strings where the first *n*–3 bits are 1's, the set of strings denoted as $\{S_{n-3}^n\}$:

   $[1,1,1,...,1,0,?,?] \equiv [u_1,u_2,u_3,...,u_{n-3},v_{n-2},?,?]$

   where the last two bits

   v_{n-1},v_n; v_{n-1},u_n; u_{n-1},v_n; u_{n-1},u_n

   are determined based on the Hasse diagram technique, i.e. are the output of the 2nPG(n') method.

...

n. We will continue this process until we arrive at the strings with the first bit 1, the set of strings denoted as $\{S_1^n\}$; meaning at the strings with two first bits

$[1,0] \equiv [u_1,v_2]$

while *n*–2 other bits

$[v_3,v_4,...,v_{n-1},v_n]$; $[v_3,v_4,...,v_{n-1},u_n]$; $[u_3,u_4,..,u_{n-1},u_n]$

are determined based on the Hasse diagram technique, i.e. are the output of the 2nPG(n') method.

In conclusion, through this code we found the general formula by which we can find the number of all strings (of length *n*) with a certain number of first bits consecutive 1's (0's):

$$|S_k^n| = 2^{n-1-k} \tag{1}$$

where $n-1$ determines the length of Hamilton's path and k the number of consecutive 1's (0's), where $1 \leq k \leq n-1$.

Based on this formula, we can find the number of strings in each step, which are:

$$|S_n^n| + |S_{n-1}^n| + |S_{n-2}^n| + |S_{n-3}^n| + \cdots + |S_k^n| + \cdots + |S_2^n| + |S_1^n| =$$

$$1 + 2^{n-1-n+1} + 2^{n-1-n+2} + 2^{n-1-n+3} + \cdots + 2^{n-1-k} + \cdots + 2^{n-1-2} + 2^{n-1-1}$$

$$= 1 + 2^0 + 2^1 + 2^2 + \cdots + 2^{n-1-k} + \cdots + 2^{n-3} + 2^{n-2}$$

Now we find the sum of these numbers which is:

$$|S_n^n| + \sum_{k=1}^{n-1} |S_k^n| = 1 + 1 \cdot \frac{1-2^{n-1}}{1-2} = 1 - 1 + 2^{n-1} = 2^{n-1}$$

If we repeat the same procedure for strings that start with a certain number of consecutive zeroes we will have the same result, 2^{n-1}. This confirms that in a total there are 2^n strings with length *n*.

Example 4.1. Determine the number of binary strings of length $n = 6$ where the first 3 bits are consecutive 1's.

Solution: According to the formula (1), we have to calculate

$$|S_k^n| = |S_3^6| = 2^{6-1-3} = 2^2 = 4$$

$$111000, 111001, 111011, 111010$$

Example 4.2. Determine the number of binary strings of length $n = 123$ where the first 17 bits are consecutive 1's.

Solution: According to the formula (1), we have to calculate

$$|S_k^n| = \left|S_{17}^{123}\right| = 2^{123-1-17} = 2^{115}$$

We can generate the required strings by recursively invoking the 2NPG(n, k) method (line 3 to 4).

Example 4.3. Determine the number of binary strings of length $n = 6$ where at least the first 3 bits are consecutive 1's.

Solution: According to the formula (1), and the fact that at least the first 3 bits are consecutive 1's, we have to calculate:

$$2|S_k^n| = 2\left|S_3^6\right| = 2 \cdot 2^2 = 8$$

$$111000, 111001, 111011, 111010, 111111, 111110, 111100, 111101.$$

Example 4.4. Determine the number of binary strings of length $n = 6$ where the first two bits are 0's while the third and fourth bits are 1's.

Solution: We apply the formula (1) for $n = 5$ and $k = 2$ so we get the number of strings that have at least the first two digits of 1's. Each of the obtained strings is added by two 0's at the beginning of the string.

$$2|S_k^n| = 2\left|S_2^5\right| = 2 \cdot 2^{5-1-2} = 8.$$

To generate the required strings, we can apply the algorithm for the $(2nPG)$ graph, obtained strings is added by two 0's at the beginning of the string.

$$0011111, 0011110, 0011100, 0011101, 0011000, 0011001, 0011011, 0011010.$$

Example 4.5. A bit string of length four is generated at random so that each of the 16 bit strings of length four is equally likely. What is the probability that it contains at least two consecutive 0's, given that its first bit is a 0? (We assume that 0 bits and 1 bits are equally likely.)

Solution: Let E be the event that a bit string of length four contains at least two consecutive 0s, and let F be the event that the first bit of a bit string of length four is a 0. We can generate the elements of events E and F by applying the $(2nPG)$ graph. Then, the probability that a bit string of length four has at least two consecutive 0 s, given that its first bit is a 0, equals

$$P(E\backslash F) = \frac{P(E \cap F)}{P(F)} = \frac{\frac{5}{16}}{\frac{8}{16}} = \frac{5}{8}.$$

5. Conclusion

This paper focuses on describing the algorithm of determining the number of binary strings which contain, for a given k, exactly k runs of 1's (0's) of length k

in all possible binary strings of length n, $1 \leq k \leq n$, the $(2nPG)$ graph algorithm has been shown to provide better results to an optimization problem when compared to an equivalent problem related to binary trees.

Detailed knowledge about the distribution of runs in binary strings may be useful in many engineering applications, for example, data compression, bus encoding techniques, computer arithmetic etc. Prior knowledge about the probability of occurrence of runs of 1's of a given length in a binary string may help us in assessing the merit of a typical run-length encoding scheme. Distribution of runs in binary strings is closely related to the statistics of success runs in n Bernoulli trials $X_1, X_2, \ldots, X_n$ with a success probability p, $0 \leq p \leq 1$ and a failure probability of $q = 1 - p$.

This paper aims to expand and even further develop the implementation of the $(2nPG)$ algorithm. This will provide many challenges that remain within our ongoing research work.

References

[1] Rosen, K.H. (2003). *Discrete Mathematics and Its Applications*. New York, NY: Mc Graw Hill.

[2] Underwood, A. (2013). Book Embeddings of Posets. Retrieved from: https://pdfs.semanticscholar.org/2d58/247171aacd351c2871a8389dd216c55aad18.pdf

[3] Cuypers, H. (2007). Discrete Mathematics. Retrieved from: https://pdfs.semanticscholar.org/707d/fd6ea8598d3715934916f73b964a979958c4.pdf

[4] Bigus, P., Tsakovski, S., Simeonov, V., Namiesnik, J., & Tobiszewski, M. (2016). Hasse diagram as a green analytical metrics tool: ranking of methods for benzo[a]pyrene determination in sediment. *Analytical and Bioanalytical Chemistry*, *408*(14), 3833-3841.

[5] Hartman, G., & Green M. (2004). Binary Strings and Graphs. Retrieved from: http://math.arizona.edu/~ura-reports/041/Green.Matthew/Final/bsgf.pdf