AN EFFICIENT REPRESENTATION FOR SOLVING
CATALAN NUMBER RELATED PROBLEMS

Matej Črepinšek[1] [§], Luka Mernik[2]

[1]Faculty of Electrical Engineering and Computer Science
University of Maribor
Smetanova Ulica 17, Maribor, 2000, SLOVENIA
e-mail: matej.crepinsek@uni-mb.si

[2]Vestavia Hills High School
2235, Lime Rock Dr., Vestavia Hills, AL 35216, USA
e-mail: lmernik@hotmail.com

Abstract: Nowadays, more and more computations, in artificial intelligence, knowledge representation, and scientific computations to name a few, require complex data processing and sophisticated algorithms, which are NP hard. Solutions to such problems might range from succinct data representations to parallelized and incremental algorithms. In this paper Catalan related problems are discussed. For efficient computation of Catalan combinations a succinct representation is used and several algorithms are developed. Results show that the suggested approach can be successfully used for solving different Catalan problems.

1. Introduction

What have the following problems in common:

— How many balanced strings of n left and n right brackets exist?

— How many full binary trees with n internal nodes exist?

§Correspondence author

— How many ways can a convex polygon with $n + 2$ sides be divided into n triangles?

— How many monotonic paths exist in $n \times n$ grid?

— How many mountain ranges can you draw with n upstrokes and n downstrokes?

Well, the solution to all of these problems is described by Catalan numbers [16]. Albeit, different problems that can be described by Catalan numbers are well studied by mathematicians, in the report [16] more than 60 different problems have been identified, there are still problems in practical use.

However, to know the number of possible solutions is only a step towards the final solution. In other words, only the size of the search space has been identified. This search space then needs to be explored (exhaustively or heuristically) by deterministic [8] or by stochastic algorithms [11] to find a particular solution. For example, in grammatical inference [7] only a small fraction of all possible derivation trees satisfy additional constraints (to parse positive samples and to reject negative samples) [6], [4]. Often search spaces are enormous, for example when their sizes are given by the Catalan numbers. With the increasing processing power of modern computers and by parallel/distributed computing many problems can be solved today which were impossible to solve ten years ago. However, to solve a problem with parallel/distributed computing, algorithms need to be carefully developed to exploit different forms of parallelism. Solutions of problems related to Catalan numbers are usually recursively defined. This prevents us from exploiting straightforward parallelism. In this paper all possible solutions (combinations) described by Catalan numbers are succinctly represented as a vector of numbers which can be computed non-recursively. The main advantage of such representation is ease of use in parallel/distributed processing and the ability that the same combination can be differently interpreted depending on the underlying problem (e.g., full binary trees, convex polygon triangulation, monotonic paths, tiling a stair step). The developed algorithms systematically examine the whole search space, without repetition of solutions, and are easy to implement and understand.

The organization of the paper is as follows. Introduction to Catalan numbers and a succinct representation of combinations are described in Section 2. Algorithms to generate the next combination from the current combination and from its index are presented in Section 3. Various transformations from this succinct representation of Catalan combinations into underlying problem representations are shown in Section 4. The paper concludes with Section 5, which summarizes the contribution of the paper.

2. Catalan Numbers and Enumeration of Combinations

The n-th Catalan number (C_n) can be calculated by the equation (see [17], [2]):

$$C_n = \frac{(2n)!}{(n+1)!n!}. \tag{1}$$

For a problem having C_n solutions to an instance of size n, we will speak of each solution as a "combination". For problem where $n = 2$ we have 2 different combinations, with $n = 4$ we already have 14 combinations, and with $n = 12$ we have 208012 different combinations. The number of different combinations for the first 12 Catalan numbers is shown in Table 1. Table 1 also describes the Catalan triangle [14], which can be used for calculating C_n.

n	C_n	n												
1	1	1	1	*1*										
2	2	2	1	2	*2*									
3	5	3	1	3	5	*5*								
4	14	4	1	4	9	14	*14*							
5	42	5	1	5	14	28	42	*42*						
6	132	6	1	6	20	48	90	132	*132*					
7	429	7	1	7	27	75	165	297	429	*429*				
8	1430	8	1	8	35	110	275	572	1001	1430	*1430*			
9	4862	9	1	9	44	154	429	1001	2002	3432	4862	*4862*		
10	16796	10	1	10	54	208	637	1638	3640	7072	11934	16796	*16796*	
11	58786	11	1	11	65	273	910	2548	6188	13260	25194	41990	58786	*58786*
12	208012	12	1	12	77	350	1260	3808	9996	23256	48450	90440	149226	208012 *208012*
⋮	⋮	⋮	⋮											

Table 1: Catalan numbers and Catalan triangle

The last number in each row of Catalan triangle (Table 1) represents C_n, where n is the row number. If we duplicate the last number (italic in Table 1), we can say that each element in the triangle is equal to the one above plus the one to the left [14].

Even for small n, the number of all possible combinations becomes large (Table 2). For example, the number of all balanced strings of 4 left and right brackets is 14, the number of all possible triangulation of octagon is 132, and the number of all possible full binary trees with 9 leaves is 1430.

From equation (1) we can calculate the number of different combinations, but equation (1) does not tell us how to systematically describe all combinations. By studying different problems a succinct representation of solutions that

can be described by Catalan numbers has been identified. The combination

$$P_n = [x_0, x_2, ..., x_j, ..., x_{n-1}]$$

represents valid solution of C_n if and only if:

1. $\forall j$; $x_j \leq j$; and
2. $\forall j$; $x_j \in \mathbb{Z}^*$; and
3. $\forall j, k$; $k > j \Rightarrow x_k \geq x_j$.

Proof. Let $P_n = [x_0, x_2, ..., x_j, ..., x_{n-1}]$ and $P_{n+1} = [x_0, x_2, ..., x_j, ..., x_{n-1}, x_n]$. According to aforemention properties $x_{n-1} \leq x_n$. If $x_{n-1} = i$ then $x_n \geq i$. Let us construct the following table

n	0	1	2	...
1	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	...
2	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	...
3	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	...
⋮	⋮	⋮	⋮	⋮

where $a_{i,j}$ denotes how many times the number j appears at position x_{i-1} in P_i. Let $a_{i,n} = 0$ if $n \geq i$ (due to the property (1)). We know that $a_{1,0} = 1$ and $a_{1,n} = 0$ if $n > 0$ (since $P_1 = [0]$ is only possibility).

There are as many $x_n = i$ in P_{n+1} as there are $x_{n-1} \leq i$ in P_n. That guarantees that all possibilities for $x_n = i$ are reached and the following equation holds $a_{i,j} = \sum_{k=0}^{j} a_{i-1,k}$. Since

$$a_{i,j} = \sum_{k=0}^{j} a_{i-1,k} = \sum_{k=0}^{j-1} a_{i-1,k} + a_{i-1,j} = a_{i,j-1} + a_{i-1,j},$$

and $a_{1,0} = 1$ the above table represents Catalan triangle ($a_{i,j}$ is the sum of the one above and the one of the left) with the property $a_{i,j} = \frac{(i-1+j)!(i-j)}{j!i!}$ for $0 \leq j \leq i$. Hence, $a_{i,i-1} = \frac{(2i-2)!1}{i!(i-1)!} = C_{i-1}$. There are the same number ($a_{i,i-1}$) of $x_{i-1} = i - 1$ in P_i as there are all combinations in P_{i-1}. Therefore there are exactly C_{i-1} combinations in P_{i-1} (or C_i in P_i). □

A similar representation can be found in many combinatorial problems [13]. All valid combinations in lexicographic order for $n = 4$ and $n = 5$ are listed in Table 2.

To be able to systematically search through combinations we enumerated combinations by index i.

To achieve our goal stated in the introductory section we need to develop

	$n = 4$						$n = 5$	
i	$combination_i$	i	$combination_i$	i	$combination_i$	i	$combination_i$
1	$[0, 0, 0, 0]$	1	$[0, 0, 0, 0, 0]$	15	$[0, 0, 1, 1, 1]$	29	$[0, 1, 1, 1, 1]$
2	$[0, 0, 0, 1]$	2	$[0, 0, 0, 0, 1]$	16	$[0, 0, 1, 1, 2]$	30	$[0, 1, 1, 1, 2]$
3	$[0, 0, 0, 2]$	3	$[0, 0, 0, 0, 2]$	17	$[0, 0, 1, 1, 3]$	31	$[0, 1, 1, 1, 3]$
4	$[0, 0, 0, 3]$	4	$[0, 0, 0, 0, 3]$	18	$[0, 0, 1, 1, 4]$	32	$[0, 1, 1, 1, 4]$
5	$[0, 0, 1, 1]$	5	$[0, 0, 0, 0, 4]$	19	$[0, 0, 1, 2, 2]$	33	$[0, 1, 1, 2, 2]$
6	$[0, 0, 1, 2]$	6	$[0, 0, 0, 1, 1]$	20	$[0, 0, 1, 2, 3]$	34	$[0, 1, 1, 2, 3]$
7	$[0, 0, 1, 3]$	7	$[0, 0, 0, 1, 2]$	21	$[0, 0, 1, 2, 4]$	35	$[0, 1, 1, 2, 4]$
8	$[0, 0, 2, 2]$	8	$[0, 0, 0, 1, 3]$	22	$[0, 0, 1, 3, 3]$	36	$[0, 1, 1, 3, 3]$
9	$[0, 0, 2, 3]$	9	$[0, 0, 0, 1, 4]$	23	$[0, 0, 1, 3, 4]$	37	$[0, 1, 1, 3, 4]$
10	$[0, 1, 1, 1]$	10	$[0, 0, 0, 2, 2]$	24	$[0, 0, 2, 2, 2]$	38	$[0, 1, 2, 2, 2]$
11	$[0, 1, 1, 2]$	11	$[0, 0, 0, 2, 3]$	25	$[0, 0, 2, 2, 3]$	39	$[0, 1, 2, 2, 3]$
12	$[0, 1, 1, 3]$	12	$[0, 0, 0, 2, 4]$	26	$[0, 0, 2, 2, 4]$	40	$[0, 1, 2, 2, 4]$
13	$[0, 1, 2, 2]$	13	$[0, 0, 0, 3, 3]$	27	$[0, 0, 2, 3, 3]$	41	$[0, 1, 2, 3, 3]$
14	$[0, 1, 2, 3]$	14	$[0, 0, 0, 3, 4]$	28	$[0, 0, 2, 3, 4]$	42	$[0, 1, 2, 3, 4]$

Table 2: Catalan combinations for $n = 4$ and $n = 5$

algorithms which can generate a valid combination from the previous combination as well as from index i. Both algorithms are presented in the next section.

3. Catalan Algorithms for the Succinct Representation

In this section two array based algorithms are presented. First, we present an algorithm which is able to produce $combination_i$ from $combination_{i-1}$. The second algorithm is able to produce $combination_i$ from i.

3.1. Setting the Next Combination

Algorithm 1 calculates the next combination from the previous one, except for the last and the first combination. The first combination is filled with zeros and labeled as $combination_1$. The data structure used to represent a combination is an array of length n. As input to the algorithm the combination P is given. As a result the next valid combination is calculated. The algorithm works by trying to change the input combination from the right to the left side and satisfy combination representation rules. It starts by setting index to the last position (line 3) and moving towards the zero position (line 16) in repeat loop. If a value at a selected index can be increased (first rule; line 5), the value is increased and all elements to the right from the current index are set to this

value (second rule; lines 7-13). After the first successful change the algorithm stops (line 14). In the case where we wish to change the last combination, the algorithm returns false.

Algorithm 1 Algorithm for calculating next combination from combination P

```
 1 function  setNextCombination (P)
 2 begin
 3    index = length (P) −1;
 4    repeat
 5      if P[ index ] < index
 6      begin
 7        inc (P[ index ]);
 8        current = index+1;
 9        while current < length (P)
10        begin
11          P[ current ] = P[ index ];
12          inc (current );
13        end;
14        return true;
15      end;
16      dec (index );
17    until index==0;
18    return false;
19 end;
```

With $n = 3$ the sequence of combinations can be calculated:
$$[0, 0, 0] \rightarrow [0, 0, 1] \rightarrow [0, 0, 2] \rightarrow [0, 1, 1] \rightarrow [0, 1, 2]$$

3.2. Creating a Combination from its Index i

In order to divide the search space into equal or nearly equal parts we need to be able to calculate $combination_i$ directly from its index i (unranking). For $n = 3$ we would like to calculate the following combinations:

$$1 \rightarrow [0, 0, 0], \qquad 2 \rightarrow [0, 0, 1],$$
$$3 \rightarrow [0, 0, 2], \qquad 4 \rightarrow [0, 1, 1], \qquad 5 \rightarrow [0, 1, 2].$$

j	4	3	2	1	0
4	1				
3	1	2			
2	1	3	5		
1	1	4	9	14	
0	1	5	14	28	42

Figure 1: Catalan triangle (CT)

Algorithm 2 Algorithm for calculating $combination_i$ from index i

---

```
1  function  createCombination (CT, n, i)
2  begin
3     int P[n];
4     dec(n);
5     pos = 0;
6     selected = 0;
7     P[pos] = selected;
8     inc(pos);
9     while pos<=n
10    begin
11       while CT[n−selected][n−pos]<i
12       begin
13          i=i−CT[n−selected][n−pos];
14          inc(selected);
15       end;
16       P[pos] = selected;
17       inc(pos);
18    end;
19    return P;
20 end;
```

To achieve this the Catalan triangle (CT) is used. The CT describes number of different combinations that can be generated by using same part of combination. For example, in Figure 1 number 42 in the column 0 and row 0 describes the fact that 42 combinations exists with schema 0**** (* means element is not defined by current column). If we move forward to the column 1 (column index in Figure 1), number 28 in row 0 describes the fact that 28 combinations exist with schema 00***, and number 14 in row 1 (column 1) describes the fact that 14 combinations exist with schema 01***. Next in column 2 schema for row

0 is 000**, in row 1 (column 2) schemas are 001** (moving left, left, up) and 011** (moving left, up, left) and in row 2 (column 2) shemas are 002** and 012**. In column 3 schema for row 0 is 0000*, in row 1 (column 3) schemas are 0001*, 0011* and 0111*, etc.

Algorithm 2 implements above logic. It generates the i-th combination from the Catalan triangle (CT) of dimension $n * n$.

The suggested algorithms are easy to implement. On the other hand, both algorithms are very efficient since memory and time consumption are small. Both algorithms, "setNextCombination" and "createCombination" have similar time complexity. At first glance both algorithms have nested loops, so complexity should be $O(n^2)$, but closer examination shows that the first algorithm, because of the return statement in the nested loop has a maximum of $(n-1) + (n-2)$ loop iterations; therefore complexity is $O(n)$. Similarly, in the second algorithm, where the nested loop is used for moving up in CT and maximum number of iterations is $(n-1)+(n-1)$, complexity is again $O(n)$. An even closer examination suggests that the first algorithm is faster than the second, because minimum iterations in the loop is 1 (in case where the most right value in combination is increased by one), and minimum of iterations in the second algorithm is $n-1$. Average calculated iterations for the first algorithm with $n \in [3..15]$ was less than 1.8 iterations and for the second algorithm less than 14.6 iterations. Space complexity in both algorithms is the same (integer array with length n), but the second algorithm has a disadvantage because in the beginning we need additional space for CT matrix (size $n \times n$). We recommend to use the first algorithm for fast segment search and the second algorithm to pick a random combination or to find a starting combination for the first one.

4. Transformation from a Combination to Underlying Problem Presentation

Since we used a single representation of Catalan combinations the next step is to show how to transform it to an underlying problem presentation. Several different problems: full binary trees, convex polygon triangulation, monotonic paths, and tiling a stair step have been selected to show that this mapping is problem specific.

4.1. Full Binary Trees

Full binary trees, also called proper binary trees, are trees in which every node has zero or two children [12]. They are used in grammatical inference as derivation trees [4], in Huffman coding for character's frequency encoding [9], in a binary decision diagram (BDD) as data structure for representing a truth table [1]. The Catalan number C_n describes number of different full binary trees with $n + 1$ leaves. Transformation from a Catalan combination into a tree is described with Algorithm 3.

Algorithm 3 Creating a full binary tree from Catalan combination P

```
1 function decodeCombinationIntoTree(P)
2 begin
3   VE = initVectorElements(length(P)+1);
4   for (j = length(P)−1; j>=0; j−−)
5   begin
6     pos = P[j];
7     newE = join(VE[pos], VE[pos+1]);
8     removeElement(VE, pos+1);
9     removeElement(VE, pos);
10    insertElement(VE, newE);
11  end;
12  return lastElement(VE);
13 end;
```

In Algorithm 3 the vector of nodes is represented as VE. In line 3 a vector of leaves is generated. In the for loop we are decoding a combination from right to left. Numbers in the combination describe positions in the vector VE. The join operator creates a new node with left and right child. After the join operation the children are replaced with the new node (lines 8-10).

An example of the decoding process for $n = 4$ and $combination_{11}$ ($[0, 1, 1, 2]$) is shown in Table 3. Every row represents one step in the decoding process. In the first and second column, the current position (j) in the combination and the combination are described. In the next column is the vector of nodes. In the first step this vector is always filled with $n + 1$ leaves. The vector position column (pos) is the current number from the combination and describes the elements to be joined. In the last two columns we can see the nodes to be joined and how the generated subtree looks like.
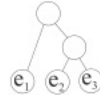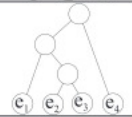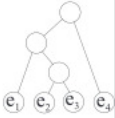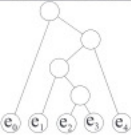
j	$Combination_{11}$	Vector of elements-VE	pos	Join	New Node
3	$[0, 1, 1, \mathbf{2}]$	$[e_0, e_1, e_2, e_3, e_4]$	2	e_2 and e_3	
2	$[0, 1, \mathbf{1}, 2]$	$[e_0, e_1, \text{(tree)}, e_4]$	1	e_1 and (tree)	
1	$[0, \mathbf{1}, 1, 2]$	$[e_0, \text{(tree)}, e_4]$	1	(tree) and e_4	
0	$[\mathbf{0}, 1, 1, 2]$	$[e_0, \text{(tree)}]$	0	e_0 and (tree)	

Table 3: Building full binary tree from $combination_{11}$

All combinations for $n = 4$ and their underlying presentation are shown in Figure 2.

4.2. Convex Polygon Triangulation

Triangulation is one of the most common problems in computer graphics [15]. The problem of finding all possible ways to divide a polygon with N sides into triangles using non-intersecting diagonals is the oldest one which is related with Catalan numbers [3]. Catalan number C_n describes the number of different ways a convex polygon with $n + 2$ sides can be cut into triangles by connecting vertices with straight lines.

The decoding algorithm from a combination into a polygon with triangulation, is similar as we described for building full binary tree in Algorithm 3. The vector of elements VE is now a vector of polygon sides. The join operator makes a new diagonal between consecutive sides. The new diagonal replaces the connecting sides. Because the last diagonal is already there, the for loop can be shorter by one.

The decoding process for $n = 4$ and $combination_{11}$ is shown in Table 4, where pos represents vector index and p side position. In the beginning we have a polygon without connecting sides. A combination describes a side connecting order. The triangulation process iterates from the outside of the polygon to the
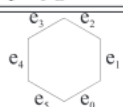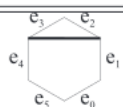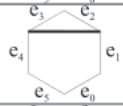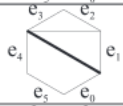
inside.
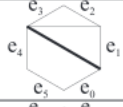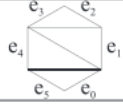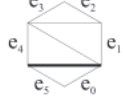
pos	$Comb_{11}$	Vector of sides - VE	Current polygon	p	Join sides	New polygon
3	$[0,1,1,\mathbf{2}]$	$[e_0, e_1, e_2, e_3, e_4]$		2	e_2 and e_3	
2	$[0,1,\mathbf{1},2]$	$[e_0, e_1, (e_2, e_3), e_4]$		1	e_1 and (e_2, e_3)	
1	$[0,\mathbf{1},1,2]$	$[e_0, (e_1, (e_2, e_3)), e_4]$		1	$(e_1, (e_2, e_3))$ and e_4	
0	$[\mathbf{0},1,1,2]$	$[e_0, ((e_1, (e_2, e_3))), e_4)]$		0	/	

Table 4: Triangulation process from $combination_{11}$

All combinations for $n = 4$ and their underlying presentation are shown in Figure 2.

4.3. Monotonic Paths on an $n \sqcap n$ Grid

In the field of path optimization different problems with monotonic paths exist, see [10], [18]. Monotonic paths have the properties that starting points are in the lower left corner, ending points are in the upper right corner, and all edges point rightwards or upwards. Catalan number C_n describes the number of different monotonic paths along the edges of a grid with $n \times n$ square cells, which do not pass above the diagonal.

The decoding algorithm from a combination into a path is very simple. Every number in a combination describes step right and its value the height of a movement. The decoding process reads the combination from the left to the right. The decoding process for $n = 4$ and $combination_{11}$ is shown in Table 4.3. The first step in the decoding process is always to move right (first row in the table). The second move can be move right or move up and right. Because we have a 1 in the second place, we move up and right. In the third step we move just right (because we are already at height 1) and in the last step we move up and right (since we have to reach height 2). At the end of decoding we need

to move to the top of the grid by simple up moves (in our case two up moves) (Table 4.3).
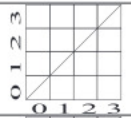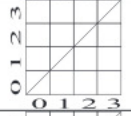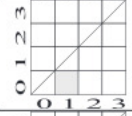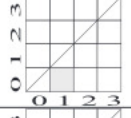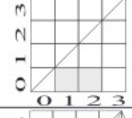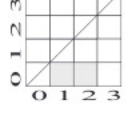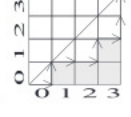
Vector index	$Combination_{11}$	Current grid	x position	y position	Grid with new path
0	$[\mathbf{0}, 1, 1, 2]$		0	0	
1	$[0, \mathbf{1}, 1, 2]$		1	1	
2	$[0, 1, \mathbf{1}, 2]$		2	1	
3	$[0, 1, 1, \mathbf{2}]$		3	2	

Table 5: Creating monotonic path from $combination_{11}$

All combinations for $n = 4$ and their underlying presentation are shown in Figure 2.

4.4. Tile a Stair Step Shape

As the last example the problem of tiling a stair step shape of height n with n rectangles is chosen [5]. The decoding algorithm from a combination into a stair step shape is similar to the one for building full binary tree in Algorithm 3. In the decoding process we have two new operations, create new tile and join tiles. The created new tile (nt) is always a rectangle. The rectangle size will be defined by its width (w) and height (h). The width is defined by size of the left child (lc) and the height by the size of the right child (rc). Child size is defined by the number of already joint elements. The size of starting elements in VE is 1. The second step is joining lc, nt and rc. First we join lc and nt, in such a way that we give nt at the top of lc (left line aligment), next we join rc at right position (top line aligment). Joining elements with size 1 is skipped.

The decoding process for $n = 4$ and $combination_{11}$ is shown in Table 4.4. All combinations for $n = 4$ and their underlying presentation are shown in

Figure 2.

j	$Comb_{11}$	Vector of elements	pos	Elements	Size w|h	New Tile	Join
3	$[0,1,1,\mathbf{2}]$	$[e_0,e_1,e_2,e_3,e_4]$	2	e_2 and e_3	1|1	e_2e_3	e_2e_3
2	$[0,1,\mathbf{1},2]$	$[e_0,e_1,\boxed{e_2e_3},e_4]$	1	e_1 and $\boxed{e_2e_3}$	1|2	e_1	$e_1\ \boxed{e_2e_3}$
1	$[0,\mathbf{1},1,2]$	$[e_0,\ \boxed{e_1\ e_2e_3},e_4]$	1	$\boxed{e_1\ e_2e_3}$ and e_4	3|1	e_4	$\boxed{e_4\ e_1\ e_2e_3}$
0	$[\mathbf{0},1,1,2]$	$[e_0,\ \boxed{e_4\ e_1\ e_2e_3}\ ]$	0	e_0 and $\boxed{e_4\ e_1\ e_2e_3}$	1|4	e_0	$\boxed{e_0\ e_4\ e_1\ e_2e_3}$

Table 6: Tile a stair step from $combination_{11}$

5. Conclusions

In this paper an efficient representation for computing Catalan combinations is presented. Every possible solution is represented as an array of numbers, with some given limitations. For searching through Catalan space, two algorithms were suggested. The first algorithm (*setNextCombination*) allows us to calculate next possible solution from the current solution without repetition of solutions. The second algorithm (*createCombination*) can create a combination from an integer in such a way that two different integers can not represent the same combination. Moreover, both algorithms are suitable for parallel/distributed processing. Catalan numbers are solutions to many interesting counting problems, such as the number of full binary trees, convex polygon triangulation, monothonic paths, and tiling a stair step. In the paper we show how our succinct representation can be mapped to all of these different problems. The presented approach was successfully implemented in the research area of grammatical inference, where we were able to achieve speedup by more than 100 times (before we used recursive algorithms) [4].

i	$Combination_i$	Full binary tree	Convex polygon triangulation	Monotonic paths	Tile stair step
1	[0,0,0,0]				
2	[0,0,0,1]				
3	[0,0,0,2]				
4	[0,0,0,3]				
5	[0,0,1,1]				
6	[0,0,1,2]				
7	[0,0,1,3]				
8	[0,0,2,2]				
9	[0,0,2,3]				
10	[0,1,1,1]				
11	[0,1,1,2]				

i	$Combination_i$	Full binary tree	Convex polygon triangulation	Monotonic paths	Tile stair step
12	[0,1,1,3]				
13	[0,1,2,2]				
14	[0,1,2,3]				

Figure 2: Catalan combinations for $n = 4$ and their domain presentation

References

[1] H.R. Andersen, An introduction to binary decision diagrams, In: *Lecture Notes for the Course Advanced Algorithms* (1997), 1-36.

[2] J.M. Borwein, D.H. Bailey, *Mathematics by Experiment: Plausible Reasoning in the 21-st Century*, A.K. Peters, Ltd., Wellesley (2004).

[3] E. Catalan, Note extraite d'une lettre adressée à l'édite, *J. Reine Angew. Mathematik*, **27** (1844).

[4] M. Črepinšek, M. Mernik, V. Žumer, Extracting grammar from programs: brute force approach, In: *Proc. SIGPLAN Notices*, **40**, No. 4 (2005), 29-38.

[5] R. Dickau, *Stairstep Interpretation of Catalan Numbers*, From the Wolfram Demonstrations Project, http://demonstrations.wolfram.com /StairstepInterpretationOfCatalanNumbers.

[6] E.M. Gold. Language identification in the limit, In: *Proc. Information and Control*, **10** (1967), 447-474.

[7] C. de la Higuera, Current trends in grammatical inference. Advances in Pattern Recognition, In: *Proc. Joint IAPR International Workshops SSPR+SPR'2000*, Springer LNCS, **1876** (2000), 28-31.

[8] J. Hromkovič, W.M. Oliva, *Algorithmics for Hard Problems*, Springer-Verlag, New York, Inc. (2002).

[9] D.A. Huffman, A method for construction of minimum redundancy codes, In: *Proc. IRE*, **4040** (1952), 1098-1101.

[10] Y.K, Hwang, N. Ahuja, Gross motion planning-a survey, In: *ACM Comput. Surv.* **24**, No. 3 (1992), 219-291.

[11] J.R. Koza, *Genetic Programming*: *On the Programming of Computers by Natural Selection*, MIT Press (1992).

[12] K. Mehlhorn, P. Sanders, *Algorithms and Data Structures*: *The Basic Toolbox*, Springer (2008).

[13] A. Nijenhuis, H.S. Wilf, *Combinatorial Algorithms for Computers and Calculators*, Academic Press (1978).

[14] F. Ruskey, *Algorithmic Solution of Two Combinatorial Problems Thesis*, Department of Applied Physics and Information Science, University of Victoria (1978).

[15] B.E. Sagan, Proper partitions of a polygon and k-Catalan numbers, http://www.citebase.org/abstract?id=oai:arXiv.org:math/0407280 (2004).

[16] R.P. Stanley, *Catalan Addedum*, http://www-math.mit.edu/r̃stan/ec/ catadd.pdf (2007).

[17] R.P. Stanley, S. Fomin, *Enumerative Combinatorics*, Volume 2, Cambridge University Press (2001).

[18] K. Sugihara, J. Smith, Genetic algorithms for adaptive motion planning of an autonomous mobile robot, In: *Tech. Rep.*, Univ. of Hawaii at Manoa (1997).