# Code-based Algorithm for Coalition Structure Generation Appendix

**Paper 1882**
**Appendix**

## A Pseudo-codes

### A.1 Pseudo-code for Constructing an Initialization

Algorithm 1 shows how ACS generates the Initializations. Given a combination of $l$ codes as input, Algorithm 1 considers these codes in order of appearance and fills the Initialization vector of size $n$ starting from agent $a_1$ (line 1) to agent $a_n$. The first loop (line 2) iterates $l$ times by processing one code at a time. On the other hand, loop 2 (line 3) fills the next $k$ elements of the Initialization vector with the code of the coalition at the index $j$ of the combination. This is formulated by $code(j)$ in line 4. The value of $k$ is defined by the size of the $j^{th}$ coalition being processed. It is expressed as $size(j)$ in line 3. To illustrate the operations of the algorithm, let us consider the node $[1, 3, 6]$ and the related combination $\{2, 0, 1\}$. The first iteration of loop 1 ($j = 0$ in line 2) considers the first code of the combination, namely 2, relating to coalition $\mathcal{C}_2$ and hence $size(0) = size(\mathcal{C}_2) = 6$. The second loop will then fill the first 6 elements ($k = 1$ to $size(j)$ in line 3) of the Initialization vector with 2, the code of $\mathcal{C}_2$ ($code(j)$ in line 4). By doing so, we assign the agents $a_1, ..., a_6$ to $\mathcal{C}_2$. The same processing is then applied to the remaining codes of the considered combination. Finally, we obtain the Initialization vector [2 2 2 2 2 2 0 1 1 1].

**Algorithm 1:** Constructing an Initialization

**Input:** A combination $\zeta$ of the codes of a node that contains $i$ coalitions.
**Output:** An Initialization vector: Init (see section 4.1).

1 $a \leftarrow 1 \triangleright a$ is a variable initialized with the index of the first agent $a_1$
2 **for** $j = 0$ *to* $i - 1$ **do**
3     **for** $k = 1$ *to* $size(j)$ **do** $\triangleright$ $size\{j\}$ returns the size of the coalition in index $j$ of the combination $\zeta$
4         Init$[a] \leftarrow code(j) \triangleright code(j)$ returns the code of the coalition in index $j$ of the combination $\zeta$
5         $a \leftarrow a + 1$
6     **end**
7 **end**
8 Return Init

### A.2 Pseudo-code for Permuting the Codes

Algorithm 2 shows how ACS generates different vectors that represent the coalition structures given an Initialization. It starts with the first agent. The first loop (line 3) iterates $i - 1$ times by considering one coalition at a time. On the other hand, loop 2 (line 5) ensures that the permutation does not occur between the codes of agents that belong to the same coalition. By doing so, we reduce the generation of duplicated coalition structures. Thus, permutation always occurs between the agent's code under processing and the agents' codes of the following coalitions. This operation is performed by the third loop in line 6. After each permutation, a new code vector is generated. Finally, by moving to the next code (agent) in line 10, we ensure that the same processing is applied to the remaining codes of the considered coalition. For clarity, the pseudo-code for evaluating the coalition structures is provided in the the pseudo-code of ACS in the paper.

**Algorithm 2:** Permuting the codes

**Input:** An Initialization Init and the number of coalitions $i$ in a node.
**Output:** A set of code vectors $\mathcal{PM}$ resulting from the permutations.

1 $a \leftarrow 1 \triangleright a$ is a variable initialized with the index of the first agent $a_1$
2 $x \leftarrow 1 \triangleright x$ denotes the index of the first agent of the next coalition
3 **for** $j = 0$ *to* $i - 2$ **do**
4     $x \leftarrow x + size(j) \triangleright$ move to the first agent of the next coalition
5     **for** $k = 1$ *to* $size(j)$ **do** $\triangleright$ $size\{j\}$ returns the size of the coalition to which the current agent belongs
6         **for** $l = x$ *to* $n$ **do**
7             Permutation $\leftarrow permute($Init$[a]$, Init$[l])$
8             add Permutation to $\mathcal{PM}$
9         **end**
10         $a \leftarrow a + 1 \triangleright$ move to the next agent
11     **end**
12 **end**
13 Return $\mathcal{PM}$

## A.3 Pseudo-code for Pruning Combinations

Algorithm 3 shows how ACS avoids redundant coalition structures. Each combination distinctly orders the coalitions. The first loop iterates over the set of combinations. For each combination in position $j$ (line 1), the second loop iterates over the rest of the combinations to test whether the order of coalition sizes is the same as in the combination in position $j$ or not. If so, it does not consider any combination that has the same coalition order with the one in position $j$. In line 4, $size(\mathcal{C}_i^j)$ returns the size of the coalition in index $i$ of the combination in position $j$.

**Algorithm 3:** Combination pruning

**Input:** A set $\vartheta$ of $m$ combinations of a partition $p$.
**Output:** A set of non redundant combinations.

1 **for** $j = 1$ *to* $m$ **do**
2     **for** $k = j + 1$ *to* $m$ **do**
3         **if** $\forall i_{/i=1..|p|}, size(\mathcal{C}_i^j) = size(\mathcal{C}_i^k)$ **then** ▷ $size\{\mathcal{C}_i^k\}$ returns the size of the coalition in index $i$ of the combination in position $k$
4             Remove the combination $k$ from $\vartheta$
5         **end**
6     **end**
7 **end**
8 Return $\vartheta$