CMST 7(1) 41-54 (2001) DOI:10.12921/cmst.2001.07.01.41-54

COMPUTATIONAL METHODS IN SCIENCE AND TECHNOLOGY 7 (1), 41-54 (2001)

COMBINATORIAL STRUCTURES IN SPIN MODELS: ALGORITHMS FOR GENERATION OF OPERATOR MATRICES

WOJCIECH FLOREK

Adam Mickiewicz University, Institute of Physics
Umultowska 85, 61-614 Poznań, Poland
e-mail: florek@spin.amu.edu.pl

Abstract. Two combinatorial algorithms, generation of ordered partitions of N with no more than m parts and decompositions of N-element set into subsets with cardinalities given by a partition $[k] = [k_0\ k_1 \ldots\ k_{m-1}]$ are presented and their possible applications to finite (mesoscopic) spin systems are indicated. The flow charts, listings, and results of test runs are provided.

1. INTRODUCTION

In the previous paper [1] a method of solving an eigen problem of a Hamiltonian acting in a finite space of states were presented. This method is based on the earlier article [2], where an expression for operator matrix elements was formulated. Application of this formula needs some group-theoretical and combinatorial structures to be introduced and then generated. Some steps can be carried out with use of standard procedures, however two of them (the generation of partitions of N with no more than m parts and the decomposition of N-element set into $lp \le m$ nonempty subsets with cardinalities $k_0 \ge k_1 \ge \ldots \ge k_{lp-1} > 0$) require new solutions - at least modifications of the standard procedures. These tasks are related, since ordered partitions $[k] = [k_0,\ k_1,\ldots,\ k_{m-1}]$ of N into m non-negative parts generated in the first step are input data in the second step. The main ideas of solutions proposed here were presented in the previous work [1], whereas this paper is devoted to more detailed discussion.

Though algorithms have been implemented in FORTRAN (F77 and F90) and C, they are presented below in the Pascal-like convention (without semi-colons), what is more readable. Some bunches of standard instructions are replaced by one command (especially the "print" command), since they are not important to the presentation of algorithms. The flow charts are also included. The last implementation and compilation were done under Linux (Red Hat 6.2) with egcs-2.91.66 on the Pentium 300 MHZ PC and the provided timings have been obtained on this machine.

2. ORDERED PARTITIONS OF N WITH NO MORE THAN m NONZERO PARTS

One of algorithms for the generation of partitions produces them in the antilexicographic order, *i.e.* a partition $[k]$ is generated before a partition $[k']$ if a word $k'_0\ k'_1 \ldots\ k'_m$, precedes lexicographically a word $k_0\ k_1 \ldots\ k_m$. It means that there exists such $0 \le l \le \min(m, m')$ that $k_i = k'_i$ for $I < l$ and $k'_l < k_l$. This algorithm makes a use of two arrays: $S[m]$ containing pairwise different parts and $R[m]$ with multiplicities (repetitions) of these parts [3,4], so it is easy to print

out partitions in the notion $[s_0^{r_0}, s_1^{r_1}, \ldots, s_{m-1}^{r_{m-1}}]$. The algorithm presented below exploits the notion of associated partitions [5], Formally, the partition $[k']$ ***associated*** with $[k]$ is determined by the following formula:

$$k_l' := \sum_{i|k_i>l} 1, \qquad 0 \le l < N.$$

Introducing the multiplicities $R[j]$ this definition can be rewritten as [5]:

$$k_l' := \sum_{i=l+1}^{N} R[i], \qquad 0 \le l < N.$$

For example, partitions $[k] = [6,4^2, 2]$ and $[k'] = [4^2, 3^2, 1^2]$ are associated, since we have:

$$k_0' = \sum_{i=0}^{3} 1 = 4, \qquad k_1' = \sum_{i=0}^{3} 1 = 4, \qquad k_2' = \sum_{i=0}^{2} 1 = 3,$$

$$k_3' = \sum_{i=0}^{2} 1 = 3, \qquad k_4' = \sum_{i=0}^{0} 1 = 1, \qquad k_5' = \sum_{i=0}^{0} 1 = 1,$$

on the one hand, and

$$k_0 = \sum_{i=1}^{16} R'[i] = 6, \qquad k_1 = \sum_{i=2}^{16} R'[i] = 4,$$

$$k_2 = \sum_{i=3}^{16} R'[i] = 4, \qquad k_3 = \sum_{i=4}^{16} R'[i] = 2,$$

on the other. In order to generate partitions with no more than *m* parts the standard algorithm is used but for each generated partitions its associated counter-part is printed out. Hence, the array *R* is preserved in the algorithm proposed (but now is called *mul*). However, for the further purposes it is more convenient to have partitions written as vectors $[k_0, k_1,\ldots, k_{m-1}]$, so the array *S* is replaced by an array *par*[*m*] and, therefore, some new modification rules are necessary. It needs introductions of two parameters, *fp* and *lp,* related to the first and the last part of the *original* (not associated) partition, respectively. During the generation these indices can reach any value from the interval [0, ***m),*** but for control purposes they can be also negative (-1, in fact). A flow chart of the proposed algorithm is presented in Fig. 1. The block "err" controls read parameters and stops the program if imposed limits are broken (too large or too small ***N*** and ***m).*** The other parts are briefly described below and their listings are presented.

In the first block all arrays and variables are initialized. It is easy to notice that the first printed partition consists of at most two different numbers: $[N/m] = jd + 1$ and $[N/m] = jd$. If $jm = N \bmod m$ than the multiplicities of these parts are *jm* and *m -jm,* respectively. Hence the associated (and generated, in fact) partition has *jd* parts equal *m* and one part equal *jm.* Therefore, in line 06 we have *mul*[0] := *jd* and the array *par* is initialized in the line 08. Since at this moment a remainder is unknown, then the first and last parts are the same (see line 07). Next the first *jm* parts have to be increased and, simultaneously, *mul*[*lp*] has to be set to 1. (Note that *mul*[0] need not to be modified). The index *lp* is determined relatively with respect to *m:* since 0 corresponds to *m*

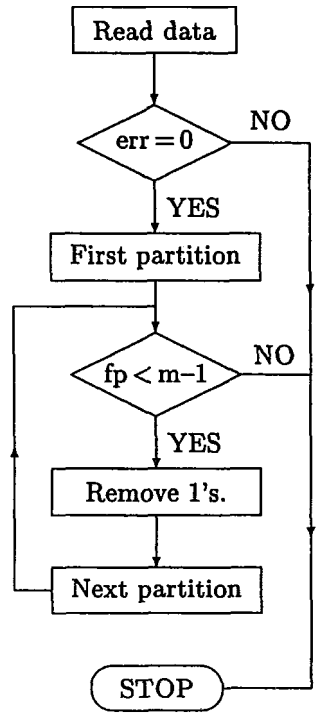

Fig. 1. Partition generation

```
01 {First partition}
02 fp := -1
03 lp := -1    {It may be ommited.}
04 jd := N div m
05 if (jd > 0) begin
06     mul[0] := jd
07     lp := fp := 0
08     for I := 0 to m - 1 do par[i] := jd
09 end
10 jm := N mod m
11 if (jm > 0) begin
12     lp := m - jm
13     if (fp < 0) fp := lp
14     mul[lp] := 1
15     jd := jd + 1
16     for} I := 0 to jm - 1 do par[i] := jd
17 end
18 print (par)
```

Fig. 2. Initialization and generation of the first partition

```
19 sum := 0
20 {Remove 1's.}
21 if (lp = m - 1) begin
22     sum := mul[m - 1]
23     par[0] := par[0] - sum
24     mul[m - 1] := 0
25     repeat lp := lp - 1 until (mul[lp] <> 0)
26 end
27 {Next partition}
28 jd := m - lp
29 sum := sum + jd
30 jd := jd - 1
31 mul[lp] := mul[lp] - 1
32 if ((lp = fp) and (mul[lp] = 0)) fp := -1
33 for I := 0 to jd do par[i] := par[i] - 1
34 lp := lp + 1
35 if (fp < 0) fp := lp
36 jm := sum div jd
37 mul[lp] := jm
38 for I := 0 to jd - 1 do par[i] := par[i] + jm
39 jm := sum mod jd
40 if (jm > 0) begin
41     lp := m - jm
42     mul[lp] := 1
43     for I := 0 to jm - 1 do par[i] = par[i] + 1
44 end
45 print (par)
```

Fig. 3. Generation of the consecutive partitions

(the first part of generated partition) then ***jm*** (the last part, respectively) corresponds to ***m - jm.*** For example, when ***N = 7*** and ***m = 3,*** then ***jd*** = 2 and ***jm*** = 1, so after the first step one obtains:

$$[\text{mul}[0]= 2, \quad par = [2, 2, 2], \quad \text{fp} = \text{lp} = 0]$$

what is modified in the second step to

$$[mul[2] = 1, \quad par = [3,2,2], \quad fp = 0, \quad lp = 2]$$

and the associated partition equals [3, 3, 1], ***i.e.*** the first (antilexicographically) partition of 7 with the first part equal to 3.

The generation of partitions ends when all parts equal 1, i.***e. [k]*** = $[1^N]$ and ***[k']*** = ***[N,*** 0,..., 0], So, in the present modification it corresponds to the condition ***fp = m*** - 1. In the other case the

next partition is generated. At first all parts equal 1 are removed. Such parts appear in [k] if ***lp = m-*** 1 or, in the other words, $k'_0 > k'_1$. Of course ***mul[m -*** 1] has to be set to zero and the new last part has to be determined. It can be done by a repeat... until loop, as in the presented listing, or by calculating a number of parts equal to ***par[***0] (it is at least two, since par[0] = ***par[***1] now). Removed 1 's and the last part ***jd = m-*** l***p*** are summed up (see the line 29) and decomposed into ***jd*** - 1 parts in a similar way as ***N*** was decomposed into ***m*** parts in the first partition. For ***N***=7 and m = 3 this algorithm generates the following eight partitions (below they are presented with their associated counter-parts):

[3,2,2]	[3,3,1]
[3, 3, 1]	[3,2,2]
[4,2,1]	[3,2,1,1]
[5, 1, 1]	[3, 1, 1, 1, 1]
[4, 3, 0]	[2,2,2,1]
[5,2, 0]	[2,2,1,1,1]
[6,1,0]	[2, 1, 1, 1, 1,1]
[7, 0, 0]	[1,1,1,1,1,1,1]

The results obtained on the Pentium 300 MHZ machine are presented in Table I. The timings are calculated as mean values of ten through million (for small values of ***N*** and ***m)*** runs of a test program without control and printing blocks. The cases when the number of partitions is greater than 2^{32} have not been tested. A number of steps necessary to generate a new partition is independent of ***N*** and m, as in the standard procedure [3,4], The visible tendency of a bit faster generation for large ***N*** and small ***m*** is caused by shorter test loops for large ***N*** and shorter substitution loops for smaller m (lines 33, 38,43).

Table I. Number of partitions *NP* and generation speed (partitions per second) for some values of *N* and *m*

N	*m*	3	4	5	6	7
100	*NP*	884	8,037	46,262	189,509	596,763
	speed	2,448,753	2,417,916	2,384,640	2,368,862	2,313,034
200	*NP*	3,434	59,823	643,287	4,775,383	26,366,879
	speed	2,464,456	2,456,156	2,450,617	2,430,671	2,413,949
400	*NP*	13,534	461,312	9,572,962	134,851,969	1,386,102,973
	speed	2,506,296	2,509,872	2,497,294	2,487,613	2,483,237
700	*NP*	41,184	2,433,337	86,994,037	2,095,277,554	$> 2^{32}$
	speed	2,511,219	2,509,872	2,505,207	2,485,834	n.a.
1000	*NP*	83,834	7,049,112	357,746,987	$> 2^{32}$	$> 2^{32}$
	speed	2,521,323	2,517,540	2,515,349	n.a.	n.a.

```
01 function subfun (k, n, a, pp): integer
02 k, n, pp, a[MaxN]: integer
03 i, p, j: integer
04 begin
05     p := pp - 1
06     if (a[k - 1] = (n - 1)) p := p - 1 else p := k - 1
07     j := a[p] - p + 1
08     for i := p to k - 1 do a[i] := i + j
09     return (p + 1)
10 end
```

Fig. 4. Modified procedure of subsets generation

3. DECOMPOSITION OF *N*-ELEMENT SET INTO *m* SUBSETS

There are some algorithms to generate k-element subsets of N-element set [3,4], however it seems they can not be modified to generate m subsets of cardinalities $k_0, k_1,..., k_{m-1}$, where $[k]$ is a partition of N. The proposed solution is based on an algorithm generating k-element subsets in the lexicographic order, ***i.e.*** it starts from {0,1,..., k-1} and goes through {0, 1,..., k-2, k}, ..., {0, 1,..., k-2, n} to {n-k, ..., n-l}. This algorithm is applied at different levels to many A'-element sets and their ¿-element subsets, so it has been included as a function. Two parameters, the cardinalities of a set and a subset, are transferred to the function "by value", whereas the other two - "by address" (the subset $a[k]$ and the so-called moving point pp). In the actual implementation p is returned as the function value. Moreover, some solutions in the main program imply that the moving point in the function has to be decreased (line 05 and **return (*p* + 1)** in line 09).

Since each partition $[k]$ of N can be represented by an ordered one, then the algorithm can be limited to this case. In the previous section ordered partitions into m non-negative parts (trailing zeroes are not truncated) has been considered, so they will be treated as the input data. Therefore, in the first step a number of non-zero parts (lp) has to be determined. The main idea of the proposed algorithm consists in decomposition of the generation process in such steps that at each moment one subset is generated with use of the standard algorithm. Let $[k] = [k_0, k_1,..., k_{m-1}]$ be an ordered partition of N into m non-zero parts. Then to generate m subsets of cardinalities $k_0, k_1,..., k_{m-1}$ one starts with k_{m-1}-element subsets of $N = N_{m-1}$, where $N_l = \sum_{i=0}^{l} k_i$ and $l = 0, 1,..., m$-1. In general, in the l^{th} step, one generates k_l-element subsets of N_l-element set, where $l = 1, 2,..., m$-1. Let us consider a partition $[k] = [3, 2, 2]$ generated by the previous algorithm. So one starts with two-element subsets of a set with seven elements 0, 1,..., 6. Of course, there are 21 subsets, {0,1}, {0,2},..., {0, 6}, {1,2},..., {4, 6}, {5,6}, and in the next steps the five-element sets will be considered: {2, 3, 4, 5, 6}, {1, 3, 4, 5, 6},..., {1, 2, 3, 4, 5}, {0, 3, 4, 5, 6},..., {0, 1, 2, 3, 5}, {0, 1, 2, 3, 4}. For each set $\{a_0, a_1,..., a_4\}$ ten two-element subsets $\{a_0, a_1\}$, $\{a_0, a_2\}$, ..., $\{a_0, a_4\}$, $\{a_1, a_2\}$, ..., $\{a_2, a_4\}$, $\{a_3, a_4\}$ are generated. So, as the final result, one obtains decomposition of the set {0, 1,..., 6} into three subsets with cardinalities 3, 2, and 2 (note that

the smallest numbers "occupy" the last subset at the start). There are 7! (3! 2! 2!) = 5 · 6 · 7 = 210 decompositions; a few of them are presented below:

{4, 5, 6} {2, 3} {0, 1}
{3, 5, 6} {2, 4} {0, 1}
{3, 4, 6} {2, 5} {0, 1}
{3, 4, 5} {2, 6} {0, 1}
. . . . . .
{0, 2, 5} {3, 4} {1, 6}
{0, 2, 4} {3, 5} {1, 6}
{0, 2, 3} {4, 5} {1, 6}
{4, 5, 6} {0, 1} {2, 3}
{1, 5, 6} {0, 4} {2, 3}
{1, 4, 6} {0, 5} {2, 3}
. . . . . .
{0, 2, 3} {1, 4} {5, 6}
{0, 1, 4} {2, 3} {5, 6}
{0, 1, 3} {2, 4} {5, 6}
{0, 1, 2} {3, 4} {5, 6}

Note that decompositions are considered as different when subsets are identical but generated in a different order, *e.g.* {4, 5, 6}, {2, 3}, {0, 1} and {4, 5, 6}, {0, 1}, {2, 3}.

Generated subsets are converted into spin configurations according with the rule: if j belongs to the i^{th} subset then a spin projection at j^{th} node is $s - l$, $l = 0, 1,..., m\text{-}1$ and $m = 2s + 1$, The listed above decompositions give rise to the following spin configurations (± stand for ±1, respectively):

$|+, +, 0, 0, -, -, -\rangle$ 2 2 1 1 0 0 0
$|+, +, 0, -, 0, -, -\rangle$ 2 2 1 0 1 0 0
$|+, +, 0, -, -, 0, -\rangle$ 2 2 1 0 0 1 0
$|+, +, 0, -, -, -, 0\rangle$ 2 2 1 0 0 0 1
. . .
$|-, +, -, 0, 0, -, +\rangle$ 0 2 0 1 1 0 2
$|-, +, -, 0, -, 0, +\rangle$ 0 2 0 1 0 1 2
$|-, +, -, -, 0, 0, +\rangle$ 0 2 0 0 1 1 2
$|0, 0, +, +, -, -, -\rangle$ 1 1 2 2 0 0 0
$|0, -, +, +, 0, -, -\rangle$ 1 0 2 2 1 0 0
$|0, -, +, +, -, 0, -\rangle$ 1 0 2 2 0 1 0
. . .
$|-, 0, -, -, 0, +, +\rangle$ 0 1 0 0 1 2 2
$|-, -, 0, 0, -, +, +\rangle$ 0 0 1 1 0 2 2
$|-, -, 0, -, 0, +, +\rangle$ 0 0 1 0 1 2 2
$|-, -, -, 0, 0, +, +\rangle$ 0 0 0 1 1 2 2

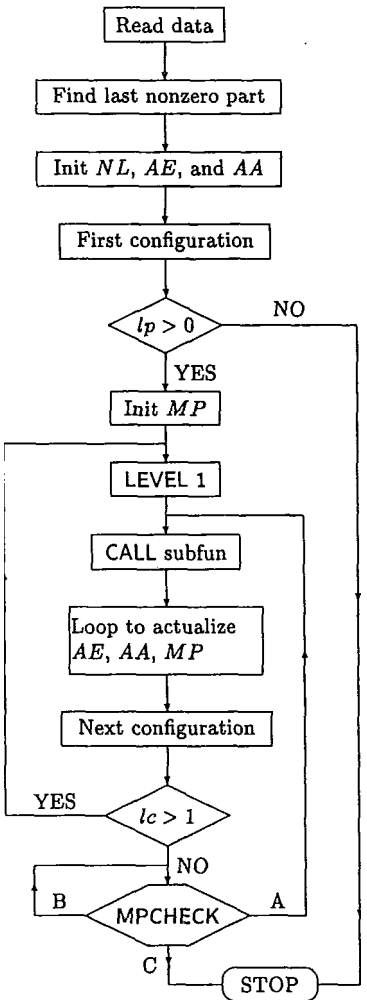

Fig. 5. Subsets (configurations) generation

In fact, to avoid negative and half-integer numbers, the program prints theses numbers increased by *s, i.e. 2s* - l, l = *0*, 1,..., *2s,* what is presented in the second column above for s=1.

The above presented remarks suggest that, first of all, there should be arrays containing generation parameters (*k, n, a, pp* in the declaration of the function "subfun") depending on the level l and the level counter l*c* = *m* - l it self. These parameters are stored in arrays *par[m], NL[m], AA[m],* and *MP[m],* where *AA* is in fact a two-index arrays since each row *AA[lc]* contains a par[lc]-element subset. However, the procedure "subfun" always works with a set {0, 1, ..., *NL[lc]* - 1}, whereas the actual elements may be different. In the above presented example this procedure generates at first two-element subsets of the set {2,..., 6}, next of the set {1, 3,..., 6} is considered, and so on till the last case - {0, 1,..., 4}. Therefore, it is necessary to introduce a two-dimensional array containing actual elements *(E)* of the set in a question. The array *par* is a part of input data and the array *NL* is determined at the start of a procedure. The other arrays

(AA, MP, E) are initialized and then modified during the generation process with use of a working array *sec[N]* containing sectioning points (it will described in more details below). The conversion from subsets to spin configurations is done by the formula

$$CONF[E[I][AA[I][j]]] = I.$$

It should be read as follows:

1. For each level i (related to spin projection $i - s$) relative indices of nodes are stored in a row $AA[i]$, $0 \le j < k_i$;
2. The i^{th} row of the array E contains absolute, *i.e.* in the range $0,\ldots, N - 1$, indices of these nodes;
3. At these nodes spin projections have the value $i - s$, so the appropriate elements of the array *CONF* equal i.

```
01 lp := n
02 repeat lp := lp - 1 until par[lp] <> 0
03 NL[0] := par[0]
04 l := N - par[0]
05 for I := 0 to NL[0] - 1 do AE[0, I] := l + I
06 for j := 1 to lp do begin
07     NL[j] := NL[j - 1] + par[j]
08     l = N - NL[j]
09     for I := 0 to NL[j] - 1 do AE[j, i] := l + i
10 end
11 for j := 0 to lp do
12     for I := 0 to par[i] - 1 do AA[j, i] := i
13 print (conf)
```

Fig. 6. Initialization and generation of the first configuration

The algorithm starts with calculation of elements $NL[j]$ as

$$NL[j] = \sum_{i=0}^{j} par[i], \qquad j = 0, 1, \ldots, lp,$$

and the initialization of *AA* and *E* in such a way, that the first decomposition into subsets is $(i = 0, 1, , \ldots, lp < m)$

$$\{N - NL[0], N - NL[0] + 1, \ldots, N - 1\},$$
$$\ldots$$
$$\{N - NL[i], N - NL[i] + 1, \ldots, N - NL[i - 1] - 1\},$$
$$\ldots$$
$$\{par[lp], par[lp] + 1, \ldots, par[lp - 1] + par[lp] - 1\},$$
$$\{0, 1, \ldots, par[lp] - 1\},$$

what corresponds to the following configuration

$$\underbrace{|lp - s, ..., lp - s,}_{\times\, par\,[lp]} \underbrace{..., i - s, ..., i - s,}_{\times\, par\,[i]} ..., \underbrace{-s, ..., -s\rangle}_{\times\, par\,[0]}$$

printed as

$$\underbrace{lp \ldots lp}_{\times\, par\,[lp]} \ldots \underbrace{i \ldots i}_{\times\, par\,[i]} \ldots \underbrace{0 \ldots 0``}_{\times\, par\,[0]}.$$

It is realized by the following starting values of $\boldsymbol{E}$ and $\boldsymbol{AA}$ (the i^{th} rows are presented):

$$AE[i] = [N - NL[i], N - NL[i] + 1, ..., N - 1, *, ..., *],$$
$$AA[i] = [0, 1, ..., k_i - 1, *, ..., *]$$

(elements denoted by an asterisk "*" are irrelevant and they need not to be initialized).

The array *MP* containing "moving points" is initialized if and only if there are more than one part. The procedure works mainly at the level $lc = 1$, *i.e.* it generates *par*[1]-element subsets of *NL*[1]-element set. Of course, at the same moment it generates *par*[0]-element subset of *NL*[0]-element set, since *par*[0] = *NL*[0]. After generation of the next subset by the function "subfun" the array *AE* has to be modified. It is done for all levels from $k = lc$ to 1 with the use of auxiliary array *sec*[*N*] constructed for each *k* as

$$sec[i] = AA[k, i], \qquad i = 0, 1, ..., par[k] - 1$$

and

$$sec[par[k]] = NL[k]$$

(note that the element *AA*[*k*, *par*[*k*]] is not determined, but, if one prefers, the array ***sec*** may not be introduced). Elements of *AE* are substituted consecutively, what is controlled by a variable *l* (lines 24-30). For a given *k* elements of *AE* at the level $k - 1$ are replaced by elements not used at the k^{th} level. To illustrate this part of the proposed procedure let us consider the previously discussed partition [3,2,2] at the moment when the level counter $lc = 2$ (for $lc = 1$ modifications of arrays are less visible). This value is reached when every tenth configurations is generated (since (5!/2! 3!) = 10). For example the 110^{th} configurations is (0 2 0 0 1 1 2) (see also the above presented example). The MPCHECK module, discussed below, leads switching to $lc = 2$ so the actual parameters of the procedure "subfun" are as follows:

$$par[2] = 2,$$
$$NL[2] = 7,$$
$$AA[2] = [1, 6],$$
$$MP[2] = 2.$$

Since the second element of *AA*[2] has reached the limit $N - 1 = 6$ then the next generated subset is

$$AA[2] = [2, 3]$$

and the moving point is changed to $MP[2] = 1$. The arrays *AE*, *AA*, and *MP* have to be modified for $k = 2, 1$. At the first case the sectioning points are 0, 2, 3, 7 (the last three are stored in the

array *sec)* and *l* is set to 0. The following subsets are considered (see lines 25, 28, and 29): {0,1}, Ø, and {4, 5, 6}. Since the pair {2, 3} is assigned to the last subset then the available elements for the next subset are 0,1, 4, 5, 6 and the row *AA*[1] is changed from [3,4] to [0, 1] (line 32). At the same moment the moving point *MP*[1] is again set to *par*[1] = 2 (line 33). Now the same

```
14 if (lp > 0) begin}
15     for (i := n - 1) downto 0 MP[i] := par[i]
16 LEV1:
17     lc := 1
18     MP[1] := par[1]

19 FUNCALL:
20     MP[lc] := subfun (par[lc], NL[lc], AA[lc], MP[lc])

21     for k := lc downto 1 do begin
22       for i := 0 to par[k] - 1 do sec[i] := AA[k, i]
23       sec[par[k]] := NL[k]
24       l := 0
25       for j := 0 to sec[0] - 1 do begin
26       AE[k - 1, l] := AE[k,j]; l := l + 1
27       end
28       for i := 0 to par[k] - 1 do
29         for j := sec[i] + 1 to sec[i + 1] - 1 do begin
30             AE[k - 1, l] := AE[k,j]; l := l + 1
31       end

32       for i := par[k - 1] - 1 downto 0 do AA[k - 1, i] := i
33       MP[k - 1] := par[k - 1]
34     end   {loop over k}
35     print (conf)
36     if (lc > 1) goto LEV1
37 MPCHECK:
38     if (MP[lc] > 1) goto FUNCALL
39     if (AA[lc, par[lc] - 1] < (NL[lc] - 1)) goto FUNCALL
40     lc := lc + 1
41     if (lc ≤ lp) goto MPCHECK
42 end   {lp > 0}
```

Fig. 7. Generation of the consecutive configurations

procedure is carried out for $k = 1$. Since the first two elements ($AA[1] = [0, 1]$) are reserved for the subset labeled by 1 then the last three (4, 5, 6) will be available for the subset labeled by 0. After this modifications one obtains:

$$AE[0] = [4, 5, 6], \quad AE[1] = [0, 1, 4, 5, 6], \quad AE[2] = [0, 1, 2, 3, 4, 5, 6];$$
$$AA[0] = [0,1,2], \quad AA[1] = [0,1], \quad AA[2] = [2,3].$$

According with (1) the generated configuration is

$$|0, 0, +, +, -, -, -> \quad \text{or} \quad 1\ 1\ 2\ 2\ 0\ 0\ 0.$$

The next call to "subfun" changes $AA[1]$ to [0, 2], so the sectioning points are 0, 2, and 5. Therefore, the available elements in $AE[0]$ are those indexed by 1, 3, and 4 in $AE[1]$, so $AE[0] = [1, 5, 6]$ and the generated configuration is

$$|0, -, +, +, 0, -, -> \quad \text{or} \quad 1\ 0\ 2\ 2\ 1\ 0\ 0.$$

The last part of the proposed procedure is devoted to control the current level. The main rule is that after generation at the level $lc > 1$ the next configuration is generated at the first level (line 36). If $lc - 1$ then some conditions are checked to decide whether all configurations at $lc = 1$ have been generated. The first condition is $MP[lc] > 1$ since $MP[lc] = 1$ (line 38) means that the end might be reached. It is checked by the next condition which is not satisfied if the last element in the row $AA[lc]$ is equal (line 39) $NL[lc] - 1$ (*cf.* the standard version of the procedure "subfun" in [3, 4]. In these cases the function "subfun" is called (the case A in the flow chart in Fig. 5). If all subsets at the level lc have been generated then lc is increased and the same conditions are checked (line 40 and 41 and the case B in Fig. 5), unless $lc > lp$, *i.e.* all levels have reached their limits (case C in the flow chart in Fig. 5).

A number of generated decompositions (configurations) is given by a polynomial coefficient $N! / \prod_{j=0}^{m-1} k_j!$, so it is very large even for small N. Moreover, the main purpose of the presented programs is applications to mesoscopic systems with $N \leq 30$. Therefore, the test runs and timings have been performed in such cases. In the test version of the algorithm the conversion (1) and printing blocks are removed. The results obtained on the Pentium 300 MHz machine are presented in Table II. The timings are calculated as mean values of several millions runs for $N = 6$ and thousands for $N = 18$. In addition, the partition [8, 8, 8] of $N = 24$ is also tested: 9,465,511,770 configurations are generated in about four hours what results in speed about 650 thousands configurations per second. In general, time necessaiy to generate one configuration increase nearly linearly with increasing N. It is confirmed by test runs for $N = 100$, where about 250 thousands configurations are generated in one second.

4. FINAL REMARKS

The combinatorial algorithms, presented in this paper, seem to work quite efficiently even for quite large values of N and m. However, both of them are limited by the "combinatorial explosion". It especially concerns the generation of decompositions into subsets (interpreted as magnetic configurations), since polynomial coefficients grow very rapidly. For example, there are more than $5 \cdot 10^{12}$ configurations corresponding to the partition [10 10 10] and their generation will take more than four months (on the Pentium 300 MHz PC). Moreover, to analyze

the ground state of an antiferromagnetic system consisting of $N = 30$ spins $s = 1$ one should consider all (non-ordered) partitions $[k_0, k_1, k_2]$ with $k_0 = k_1$[1]. Furthermore, this step is only the beginning of a long process: configurations have to be stored and decomposed into orbits, matrix elements of operator matrices have to be determined and eigenproblems solved. On the other hand, systems with $s = 1$ and $N \leq 20$ lead to several millions configurations for each partition (133,024,320 for [7 7 6]), so the generation process takes less than five minutes (assuming $0.5 \cdot 10^6$ configurations per second). Each configuration $|m_0, m_1,..., m_{N-1})$ can be stored as a number $a = \Sigma_{j=0}^{N-1} (m_j + s) \cdot (2s + 1)^j$, *i.e.* a configuration is considered as a number written in the base $(2s + 1)$. In the presented example, $N = 20$ and $s = 1$, the maximal value of a equals $3^{20} - 1 = 3{,}486{,}784{,}400 < 2^{32} - 1$, so one needs four bytes per configurations, what results in more than 500 MB for the partition [7 7 6]. It is a very large number, but all configurations can be stored on a hard disk used in PC's. It seems, however, that the reasonable limit is $N = 18$, where number of configurations is about seven times smaller (e.g. 17,153,136 for [6 6 6]). Larger spin values decrease this limit: a number of configurations reaches $15 \cdot 10^6$ for $N = 15$ when $s = 3/2$ and so on, but $4^{15} < 2$ so again four bytes is enough to store one configuration. Such limits allow to investigate most of mesoscopic magnetic rings [6], what is the main purpose of the presented algorithms.

Table II. Number of configurations *NC* and generation speed (configurations per second) for $N = 6$ and $N = 18$

$N = 6$					
[*k*]	[5 1]	[4 2]	[4 1 1]	[3 3]	[3 2 1]
NC	6	15	30	20	60
speed	1,597,444	1,348,072	1,390,691	1,315,011	1,323,714
[*k*]	[3 1 1 1]	[2 2 2]	[2 2 1 1]	[2 1 1 1 1]	[1 1 1 1 1 1]
NC	120	90	180	360	720
speed	1,428,163	1,435,933	1,401,214	1,407,239	1,364,670
$N = 18$					
[*k*]	[9 9 0]	[9 8 1]	[9 7 2]	[9 6 3]	[9 5 4]
NC	48,620	437,580	1,750,320	4,084,080	6,126,120
speed	596,272	623,760	653,922	688,761	789,093
[*k*]	[1 2 6 0]	[1 2 5 1]	[1 2 4 2]	[1 2 3 3]	[1 6 1 1]
NC	18,564	111,384	278,460	371,280	306
speed	632,461	716,757	770,845	832,466	849,292

The problem of generation all decompositions of *N*-element set into subsets with cardinalities given by an ordered partition $[k_0\, k_1, \ldots\ k_{m-1}]$ has been solved in such a way that there is no recurrence, since sometimes it is difficult to realize recurrent algorithms. It should be underlined that in the proposed procedure all k_1-element subsets of *NL*[l]-element subset are generated many times (strictly speaking - $N! / (k_0 + k_1)! \prod_{j=2}^{m-1} k_j!$ times). To avoid this feature one can store all

decompositions of $NL[1]$-element set into k_0- and k_1-element subsets and then use stored results for the further generations at levels $lc > 1$. However, it give rise to a problem of efficient storing and recalling of configurations. Moreover, the actual form of the algorithm is more flexible: the array AE may contain any elements (for example, an element $AE[i][j]$ can be interpreted as an index of an array with actual objects in question) or the generation can be limited to subsets with given properties. It is worth noting that consecutively generated decompositions may differ considerably, *e.g.* {0 2 3} {4 5} {1 6} and {4 5 6} {0 1} {2 3} in the example presented in the main text. It seems interesting to construct an algorithm which generates decompositions with the minimal difference, when consecutive decompositions differ only by one interchange of two elements between two subsets, *i.e.* a decomposition in a form

$$\{\ldots\}\ \ldots\{\ldots a\ldots\}\ \ldots\ \{\ldots b\ldots\}\ \ldots\ \{\ldots\}$$

is always followed by a decomposition obtained by swapping a and b only:

$$\{\ldots\}\ \ldots\ \{\ldots b\ldots\}\ \ldots\ \{\ldots a\ldots\}\ \ldots\ \{\ldots\}.$$

Such algorithm is known for $m = 2$ [3,4], *i.e.* for generation of k-element subsets.

Acknowledgments

This work is supported by the Polish State Committee for Scientific Research (KBN) within the project No. 8 T11F 027 16. The author is indebted to Prof. G. Kamieniarz for inspiring and valuable discussions. Some of numerical calculations were carried out on SGI Power Challenge in the Supercomputing and Networking Center in Poznań.

References

[1] S. Bucikiewicz, W. Florek, CMST, this volume.

[2] G. Kamieniarz, R. Matysiak, W. Florek, S. Walcerz, J. Magn. Magn. Mater., **203**, 271 (1999).

[3] E. N. Reingold, J. Nivergelt, N. Deo, *Combinatorial Algorithms. Theory and Practice,* Prentice Hall, Engelwood Cliffs NJ, 1977.

[4] W. Lipski, *Combinatorics for Programmers.* Polish Sci. Publ. (PWN), Warsaw, 1982 [in Polish].

[5] A. Kerber, *Algebraic Combinatorics via Finite Group Actions.* B I Wissenshaftsverlag, Mannheim-Wien-Zürich, 1991.

[6] D. Gatteschi, A. Caneschi, L. Pardi, R. Sessoli, Science, **265**, 1054 (1994); A. Lascialfari, D. Gatteschi, A. Cornia, U. Balucani, M. G. Pini, A. Rettori, Phys. Rev. B, **57**, 1115 (1998); A. Caneschi, D. Gatteschi, C. Sangregorio, R. Sessoli, L. Sorace, A. Cornia, M. A. Novak, C. Paulsen, W. Wernsdorfer, J. Magn. Magn. Mater., **200**, 182 (1999).