The 7th Balkan Conference on Operational Research
"BACOR 05"
Constanta, May 2005, Romania

A METHOD FOR THE SINGLE DIRECTION NESTING COMPUTATION

CRISTIAN ŞTEFAN DUMITRIU
Utilnavorep S.A.

ALINA BĂRBULESCU
"Ovidius" University of Constanta, Romania

Abstract
In this paper we present a simple nesting optimization method for a single dimension direction. The method consists in sequential generations of certain combinations searching for fine results using reasonable computation resources. The algorithm has been transposed in Visual Basic for checks and exemplifications. The results consist in the exact sequence of the given pieces along the row material bar and indications regarding computation volume for the initial optimization conditions.

Keywords*: nesting, length of bar, combination, piece*

1. INTRODUCTION

These days, the latest technology in construction machineries and computation resources significantly increased the productivity and goods quality. In these conditions the production activity request quick processing and analysis methods of the technological information. For instance, the high number of the structure pieces in ships building generates difficulties in preparing of cutting information and sensible looses of materials (and money in case of expensive materials) where improper distribution of the pieces into the row materials exists. The specialized software for the projects development usually include modules for optimization of the row materials consumption, but those programs imply high expenses.

In what follows we shall present a program conceived by us, in order to solve the problems related to the simple nesting.

This program is made to generate and select a set of combination that has all the pieces selected only once into the entire set. The program can find this set of combinations in two ways. The first method is the finest way and represents a full

generation, restricted by the length of row material. After the process of generation, the combinations are decreasing sorted and then selected.

In the same conditions as in the first case, in the second one, fewer combinations will be generated, according to a constant named "allowed loose per bar", given by the user. Choosing the second way, the generation time dramatically decreases, but the combination results may differ from the first case.

Before the explanation of the generation mechanism, we shall define the "length of bar" condition.

The given length of the bar is considered to be an interval between the length of the bar minus the "allowed loose per bar" value and the length of the bar (row material). This means that for an accepted full bar filled combination the length of the combination (the sum of the pieces lengths) should be into the above explained interval.

During the generation if a certain combination satisfies the "length of bar" condition, the pieces used in that combination are recorded only to the final combination files. If this generation is missing from the current temporary file, the algorithm will be absolved from the generation of new possible combinations based on this one, even if the new ones has the sum of lengths higher than the source (the missing combination). If the "allowed loose per bar" value is equal to zero or is lower than the value resulted as the difference between the given length of bar and any sum of combinations length, the results of generation will be the same as in the first method.

2. THE NESTING PROGRAM

The algorithm is divided in 6 parts as follows:

a. Loading the data as: the length of the nesting length bar (LB), the number of pieces (NP), the length of each piece (LPi), the allowed loose per bar (AL) and the technologically loose per piece (TL). Each length of piece is calculated as the sum LPi + TL and is compared with LB. The value LBmin used in code is calculated as LB – AL.

b. Setting the output format. During the generation, the nesting information is manipulated by six files written directly to hard disc.

Three files, denoted by #1, #3 and #5, keep the combination string, like "1546825" and the other three, denoted by #2, #4 and #5, keep the value (sum of the pieces values) of the combinations.

The files #1, #2, #3 and #4 are used for combination step by step and the files #5 and #6 store the final information regarding the generated combinations.

Depending on the number of pieces, the combination string is built from numbers like 1, 122, 13…, resulting a string like "1122<u>13</u>…". This information cannot be used because is impossible to determinate which pieces composed the above string.

In order to normalize the string length, the information about each combination is stored at the length = (the number of pieces) × (the digits number of the number of pieces), resulting a string like "<u>001</u>122<u>013</u>…". Using this format, we can easily find how many pieces are in string (for the above string is: 9 digits / 3 digits) and which pieces have been combined in string.

c. Initialization. The combinations are generated in a cascade method. This means that the new combinations are generated based on the previous ones. So, at the first step the combinations are exactly the pieces, one by one.

This section of the generation writes the number of pieces from 1 to N to the files #1 and #5.

d. *The combinations generation.* The total number of combinations (TC) generated for "N" numbers is given by the formula $TC = 2^N$. Since we have a restriction for the correspondent sum of the pieces combined, it is obviously that some combinations should not be generated and stored because their sum exceed the length of bars, LB.

In order to simplify the explanation and to see exactly how the combinations are made, we shall analyze an example with the following initial data:

- the number of pieces, NP = 5, and the pieces dimensions (LPi) 30, 50, 10, 20 and 60
- the length of the bar to be filled with pieces, LB = 100
- the maximum allowed loose per bar, AL = 10.

For five pieces, the total number of combination is $2^5 = 32$. Excluding the combination C_N^0, 31 combinations remain.

Reading the file #1 we shall find N numbers, from 1 to N. Those numbers will be composed one by one with the next numbers that succeed it in the generation target. For example, the number "1" will be combined with the numbers "2", "3", "4" and "5" and the combinations "12", "13", "14" and "15" will be obtained.

The next generation of combinations uses the previous one, for instance "12". From this one, the last character of the combination string is identified – in this case, "2" - and it is combined with the numbers that follow it, "3", "4" and "5". The new combinations are "123", "124" and "125". Having N pieces, the combinations will be generated in N-1 steps (the last piece cannot be combined with another one), called in the code, STEP.

The complete generation is shown in the chart 2.1.

Since the next step is based on the previous one, it is necessary to count the number of all the combinations made at each step, called in the code, LINES. For the first STEP, the variable LINES is equal to N.

The generated combination and the value of the correspondent sum are written to files #3, #5, #4 and #6. In order to reduce the volume of the processing time and the files size, it is necessary to compare the sum value of combination with LBmin. If the sum value is lower than the LBmin, the information of the combinations will be written to the previous files.

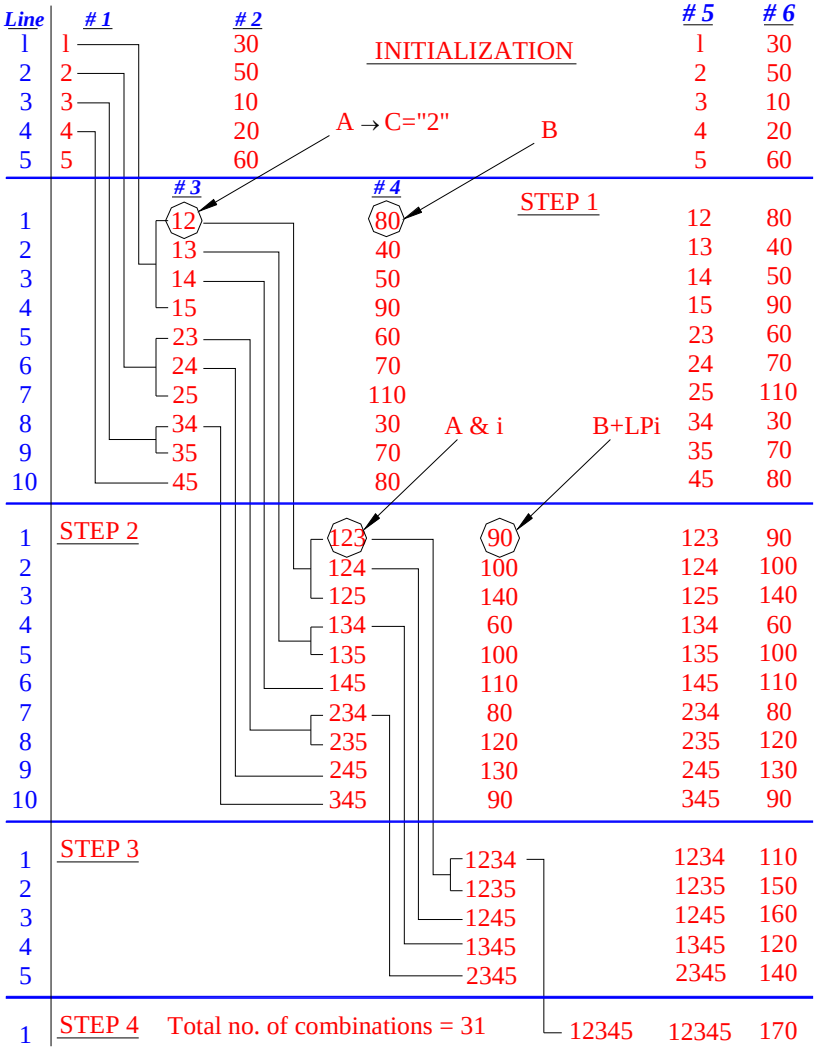

Chart 2.1 The schema of the complete generation for five pieces

After finishing the current STEP generation, the files #3 and #4 used to write information will be renamed as file #1 and #2, in order to be used to read information for a new generation.

At this moment, we found all the valid combinations that satisfy the requirements of our example generated under condition of LB.

The code switches the generation to the second method if the condition of the previous check is not reached but the sum values is less than LB. This case is possible only if the value of REST is bigger than 0. Each combination made by this method is written only to the files #5 and #6. The generation results for LB condition and LBmin

condition when AL = 0 is 20 combinations instead of 31, as it was generated before. This generation is shown in chart 2.2.

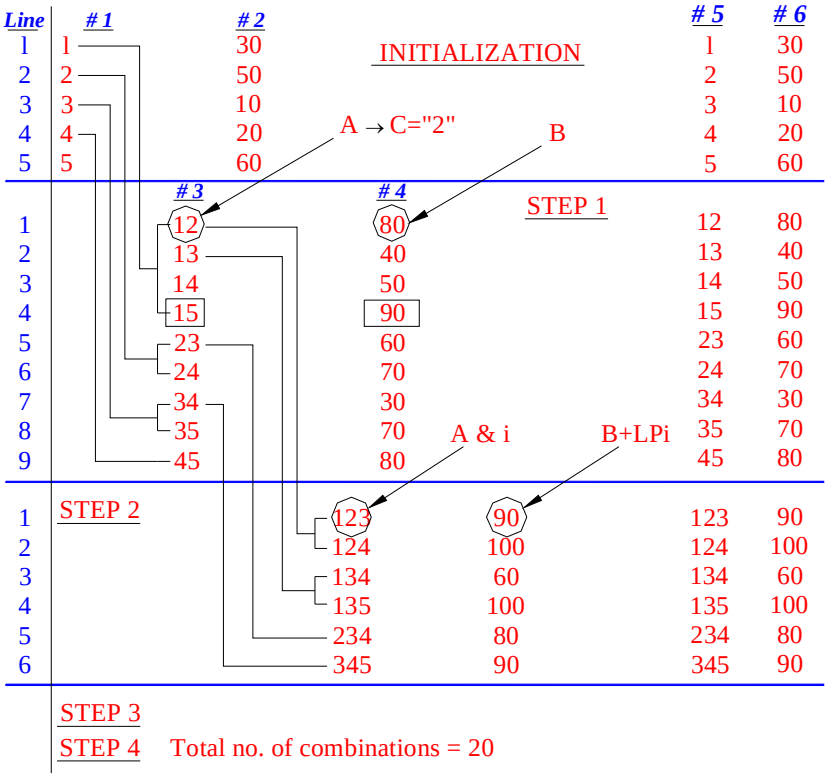

Chart 2.2 The schema of the valid combination

A schematic code for the combination generation is written below.

```
For STEP = 1 To NP-1
        NEWLINE
        For LINE = 1 To LINES
                If #1 "still have records to be read" Then
                        Read A and B
                        Set C = Right (A, d)
                        If C < NP Then
                                For i = C + 1 to NP
                                        SUM = B + LPi
                                        If SUM < LBmin Then
                                                Print A&i to #3 and #5
                                                Print SUM to #4 and #6
                                        Elseif  SUM ≤ LBmin Then
                                                Print A&i to #5
```

```
                                        Print SUM to #6
                                        GOTO NEWLINE
                                    Else: End if
                                Next i
                            End If
                        End if
                Next LINE
                Rename #3 as #1
                Rename #5 as #3
Next STEP
```

e. Sorting the combinations. The best combination is the combination for which the sum of the lengths is closer to the length of the row material. This is the reason for which the result of the combination should be sorted decreasing. The used sort method is a worldwide known method named "Shell Sort". The processes of sorting use the computer RAM memory.

f. Selecting the combinations. After sorting, the first combination in the list of the combinations is chosen. Its combination numbers are stored into the vector of used pieces, denoted by VP. The next combination selected will be the next one from the list of combination that does not contain any number from the VP. The selection continues until the counted numbers in the VP is equal to N.

For exemplification of the two methods we have chosen 40 pieces that have been combined by each method. The results are shown in the followings standard generation reports.

Case no. 1 – full generation method where allowed loose per bar = 0
INPUT DATA
No. of generation pieces = 40
No./Length
1 / 30; 2 / 50; 3 / 10; 4 / 20; 5 / 60; 6 / 30; 7 / 5; 8 / 10; 9 / 20; 10 / 60; 11 / 14; 12 / 50; 13 / 10; 14 / 20; 15 / 10; 16 / 30; 17 / 25; 18 / 10; 19 / 15; 20 / 60; 21 / 30; 22 / 50; 23 / 10; 24 / 45; 25 / 35; 26 / 30; 27 / 23; 28 / 10; 29 / 78; 30 / 2; 31 / 10; 32 / 20; 33 / 10; 34 / 30; 35 / 50; 36 / 10; 37 15; 38 / 60; 39 / 30; 40 / 50

Nesting length bar = 100
Allowed loose per bar = 0
Technologically loose = 0

GENERATION RESULTS
Selections from 377455 combinations

Bar no. 1 : REST = 0 --> 4 + 6 + 7 + 18 + 23 + 36 + 37 = 100
Bar no. 2 : REST = 0 --> 3 + 9 + 16 + 28 + 31 + 32 = 100
Bar no. 3 : REST = 0 --> 8 + 13 + 14 + 15 + 17 + 27 + 30 = 100
Bar no. 4 : REST = 0 --> 2 + 12 = 100

Bar no. 5 : REST = 0 --> 22 + 35 = 100
Bar no. 6 : REST = 0 --> 1 + 33 + 38 = 100
Bar no. 7 : REST = 1 --> 11 + 25 + 40 = 99
Bar no. 8 : REST = 7 --> 19 + 29 = 93
Bar no. 9 : REST = 10 --> 20 + 39 = 90
Bar no. 10 : REST = 10 --> 5 + 34 = 90
Bar no. 11 : REST = 10 --> 10 + 26 = 90
Bar no. 12 : REST = 25 --> 21 + 24 = 75

Process duration times :
- generation = 37 sec
- sorting = 366 sec
- selecting = 14 sec

TOTAL DURATION = 417 sec

Case no. 2 – fast generation method where allowed loose per bar > 0

The number of pieces and lengths are identically with those from the case no. 1.

….

Nesting length bar = 100
Allowed loose per bar = 10
Technologically loose = 0

GENERATION RESULTS
Selections from 53062 combinations

Bar no. 1 : REST = 0 --> 7 + 15 + 18 + 23 + 31 + 34 + 36 + 37 = 100
Bar no. 2 : REST = 0 --> 4 + 14 + 20 = 100
Bar no. 3 : REST = 0 --> 2 + 12 = 100
Bar no. 4 : REST = 0 --> 3 + 16 + 19 + 24 = 100
Bar no. 5 : REST = 0 --> 32 + 39 + 40 = 100
Bar no. 6 : REST = 0 --> 21 + 28 + 33 + 35 = 100
Bar no. 7 : REST = 0 --> 6 + 8 + 10 = 100
Bar no. 8 : REST = 0 --> 9 + 22 + 26 = 100
Bar no. 9 : REST = 1 --> 11 + 27 + 30 + 38 = 99
Bar no. 10 : REST = 10 --> 1 + 5 = 90
Bar no. 11 : REST = 12 --> 13 + 29 = 88
Bar no. 12 : REST = 40 --> 17 + 25 = 60

Process duration times :
- generation = 5 sec
- sorting = 32 sec
- selecting = 2 sec

TOTAL DURATION = 39 sec

3. CONCLUSIONS AND REMARKS

The table 3.1 contains the results of six cases of combinations generation made on two computers with 400 MHz and 2600 processor frequency.

Comparing the cases 3 and 4 we can conclude that the full generation (first method, allowed loose per bar is equal to zero) is not desired in the case of many pieces, due to the number of combination and implicit time of processing.

As we can see, the number of combination made depends on many factors, like the number of pieces, the average length and the distribution of the pieces (not exemplified here).

Data	1	2	3	4	5	6
Processor frequency	400 MHz					2.6 GHz
N (no. of pieces)	30	30	40	40	50	50
Average length	~ 28.4				~ 28.1	
LB (length of bar)	100					
AL (allowed loose)	0	10	0	10	10	10
Combinations made	53,169	9,243	377,455	53,062	283,027	
Total time min:sec	0:37	0:6	6:57	0:39	5:01	0:54

Table 3.1 Comparison of the nesting time

Since the generation is made from the first piece to the last one, a new arrangement of pieces will generate a different result for the number of combinations generated or for the processing time. In this situation we can believe that arranging the pieces before starting of generation may advantage us. For example, we can sort decreasing or arrange the longest pieces near to the shortest ones and the middle lengths to the end. These tricks can be studied and come into the practice user or can be implemented into the program as options and recommendation where possible.

Analyzing the report processing times for both exemplified cases we find that the sorting time is few times longer than the generation time. This fact requests to change the sorting method to reduce the total generation time that may allow increasing the number of generation pieces.

Basically, this program combines and arranges many pieces with different lengths on other pieces with a fixed length. This kind of work is often made in metallic or other confections but is not the only one performed. Another work request, of same importance, is the arrangement of bi-dimensional pieces, like thin steel plates or rectangular pieces, into standard steel plate formats. We think that starting from the methods used for single direction nesting it is possible to make a useful program for two direction nesting.

BIBLIOGRAPHY

[1] Tomescu, Ioan (1981), “Problems of combinatory and graph theory”, E.D.P, Bucuresti (in Romanian);

[2] Schneider, David (1999), “An introduction to Programming Using Visual Basic 6.0”, 4th Edition, Prentice Hall.