Modular enumeration - a new method for solving discrete programming problems

Vitaly O Groppen¹
¹North-Caucasian Institute of Mining and Metallurgy (State Technological University), Nikolajev str. 44, Vladikavkaz, 362021, Russia

Abstract. A new approach resulting in a new group of algorithms for finding globally optimal solutions to discrete programming problems with Boolean variables is proposed. It is proved that the effectiveness of this method depends only on the number of variables of the problem to be solved and of the amount of RAM of used computer. The main difference from the existing methods of implicit enumeration is in the ability for some of the modular enumeration procedures to a priori predict, solving any extreme problem with Boolean variables, the number of iterations, the gain in running time in comparison with the traditional enumeration scheme, as well as the amount of RAM used. The experimental results confirm high efficiency of the proposed approaches.

1. Introduction

The efforts associated with the normal tendency to reduce the running time when solving various applied problems can be identified as analytical and computer-based [1] – [4]. The latter dominate in relation to extreme problems described by discrete mathematical models, where search for globally optimal solutions in the general case relies on various enumeration procedures. In this case, as a rule, either complete enumeration methods or implicit enumeration methods developed in the middle of the last century are used, such as dynamic programming, branch-and-bound (B&B) methods, backtracking [5] – [9] and their modifications [10] – [14]. Keeping in mind all the positive features of these methods, it should be recognized nevertheless that all of them have several drawbacks:

• the impossibility to predict a priori neither the number of iterations, nor its upper bound and, as a result, it is impossible to predict the gain in this number from the use of these procedures as compared to the complete enumeration methods;
• operating in a hostile environment, that is in the case when the B&B method, as well as backtracking, makes many mistakes when choosing the direction of search in the branch tree, there may be cases when the number of iterations by these methods exceeds the number of different complete plans, which means that running time of implicit enumeration methods sometimes exceeds the running time of the full enumeration method;
• using dynamic programming the number of analyzed plans can also exceed that of different complete plans - it takes place when, searching a globally optimal solution, it is possible to cut off only a relatively few unpromising groups of variables’ vectors due to the specifics of the being solved problem.

Meanwhile, any complete enumeration method makes it possible to predict a priori the number of iterations for globally optimal solutions search to specific problems. Another advantage of complete enumeration is the reduced requirements for the amount of used RAM. As the only drawback of this method is the relatively long runtime, the idea of improving the method of complete enumeration of all combinations of values of variables in relation to discrete optimization problems remains relevant.
The purpose of this work, developing the ideas outlined in [15], is to create a new class of enumeration algorithms, which allow:

a) to reduce the running time of the latter;
b) to adapt the algorithm to the size of the computer's RAM, whereas the goal remains the same - to minimize the computation time;
c) to preserve a possibility of an a priori estimate of the number of iterations in relation to a particular problem to be solved.

The main idea of the algorithms proposed below is in the modular organization of the exhaustive search, which makes it possible to minimize repetitive computations at each iteration by storing their results in RAM and using them as needed.

This paper contains description, analysis, and experimental verification of efficiency of the three new realizations of the complete enumeration method including its modular organization related to searching a globally optimal solution to extreme problems with Boolean variables. Examples and statistics below are based on knapsack problems solving [14].

2. Basic Ideas of Modular Enumeration

The essence of the proposed approach is to implement three stages of finding globally optimal solutions to discrete programming problems. At the first stage, all the variables are grouped into “m” modules and, for each module, all combinations of the values of the variables corresponding to this module are generated and stored in the RAM. At the second stage, for each combination of variables of each module, its part of the objective function and constraints are calculated and stored.

The third stage includes two steps:

a) use of the contents of the modules for generation of all the complete plans, corresponding values of the objective function and constraints;
b) their comparison and selection of the best values of the vector of variables.

Below it will be shown how, by modifying these steps, modular enumeration can be adapted to the specificity of various tasks. Further, we assume that these algorithms are solving problems with Boolean variables of the form (see [11] – [13]):

\[
F = \sum_i^m C_i z_i \rightarrow \max(min); \\
\sum_i b_{ij} z_i \leq a_i; \quad j = 1, 2, \ldots d; \\
\forall i: z_i = 1, 0,
\]

where \( Z = \{z_1, z_2, \ldots, z_n\} \) is a vector of Boolean variables, \( n \) – their number, whereas \( C_i, b_{ij}, a_j \) are integer constants.

It can be shown that the running time \( T_1 \) of the complete enumeration search for problem (1) globally optimal solution can be determined by the following expression:

\[ T_1 = (d+1)(n-1)2^n t_0, \]

where \( t_0 \) is equal to the sum of the times of addition and multiplication of two integer numbers, whereas the time of two numbers comparison is ignored.

If the modular enumeration is used to solve problem (1), regardless of the number of variables of this problem, the running time of the 3rd stage of this algorithm does not exceed the \( T_2 \) value:

\[ T_2 = (d+1)(m-1)2^n t_0; \]

\[ k \cdot m \cdot 2^{n_m} \leq M, \]

where “m” is the number of modules \((m > 1)\), \( n_p \) is the number of variables, belonging to the p-th module, \( k \) is the proportionality coefficient, \( M \) is the amount of free RAM in the computer used.

This poses three questions:

- What is the optimal number of modules for a specific task?
• What is the gain in running time for finding problem (1) globally optimal solution when applying modular enumeration if compared to the traditional enumeration scheme?

• What should be the strategy of variables distribution between modules?

From (3) it follows that the minimum value of $T_2$ corresponds to the minimum value of $m$ determined according to (4). Obviously, with a sufficiently large RAM size, the optimal value of $m$ is equal to two. The latter makes it possible to determine the upper bound for the gain $\eta$ in the time spent on the finding of a globally optimal solution to problem (1) when using modular enumeration in comparison with the traditional enumeration scheme. Considering that the minimum value of $m$ is equal to two and ignoring the second term in square brackets in (3), the ratio of the right-hand sides (2) and (3) results in the following equality:

$$\eta = \frac{T_1}{T_2} = n-1.$$  \hspace{1cm} (5)

In other words, according to (5), upper bound for the gain in the running time $\eta$ does not depend on the number of constraints $d$ in (1) and is uniquely determined by the number of variables in this problem. Since when deriving (5), the first two mentioned above stages of searching for system (1) optimal solution by modular enumeration were ignored, associated with them time spent on generating and filling the modules was also ignored, that is why the value of $\eta$ in (5) can be considered as the upper bound of the gain in computation time. Moreover, considering that, in the first approximation, the time spent by modular enumeration on these two stages can be described as: $T_3 = t_0 \cdot n \cdot 2^{0.5n}$, the following expression is true:

$$\lim_{m \to \infty} \frac{T_3}{T_2} = 0.$$  \hspace{1cm} (6)

Since due to (3) - (4) system optimal number of modules in the modular enumeration algorithms for any amount of free RAM in the computer used can be determined, the problem arises of the optimal distribution of variables between the modules. This problem is solved by the following theorem:

Theorem 1. For the number of variables of the problem (1) "n", which is a multiple of the number of modules "m", the optimal is a uniform distribution of variables in these modules.

Proof: The running time $T_1$ corresponding to the search by modular enumeration algorithm with the uniform distribution of variables between the modules, i. e. $\forall 1 \leq p \leq n/m$: $n_p = n/m$ from (3) it follows:

$$T_1 = t_0 \cdot (d-1) \cdot [n \cdot 2^{n/m} + (m-1) \cdot 2^g].$$  \hspace{1cm} (7)

We obtain a new, non-uniform distribution of variables between the modules, for which in the uniform distribution we remove $g$ variables ($0 < g < n/m$) from one module and transfer them to the other module. The search time for problem (1) solution in this case is denoted as $T_2$:

$$T_2 = t_0 \cdot (d-1) \cdot \left( \frac{n}{m} \cdot (m - 2) \cdot 2^{n/m} + \left( \frac{n}{m} - g \right) \cdot 2^{n/m-g} + \left( \frac{n}{m} + g \right) \cdot 2^{n/m+g} + (m-1) \cdot 2^n \right).$$  \hspace{1cm} (8)

The difference $T_2 - T_1$ is designated bellow as $\Delta T$. After expanding the parentheses and transformations, we get:

$$\Delta T = t_0 \cdot (d-1) \cdot 2^{n/m} \cdot \left( [2^g + 2^g - 2] + 2^g \cdot (2^g - 2^g) \right).$$  \hspace{1cm} (9)

It is easy to verify that for any integer values of $g$ in the range $[1 - n/m]$, the right-hand side of (9) is non-negative. It follows that $T_1 < T_2$. The theorem 1 is proved.

3. Modular Enumeration Algorithms

Modular enumeration is a paradigm that allows you to generate various enumeration algorithms. Three of these algorithms are presented and further explored below.

The main feature of Algorithm 1 presented below is such an organization of the full enumeration process, in which:

a) the conditions of Theorem 1 are satisfied;

b) amount of free RAM in the used computer satisfies (4) thus resulting in $m = 2$;

c) the number of arithmetic operations is minimal.

Algorithm 1

Step 1. If problem (1) goal function $F$ is maximized, then $R = -\infty$, otherwise $R = +\infty$. 

3
Step 2. The entire set of variables Z is divided into \( m = 2 \) modules, the number of variables of the first module is equal to the integer part of the ratio \(| Z | / 2\), the number of variables of the second module is equal to \(| Z | - | Z | / 2\), subsets of variables of these modules do not intersect, in other words, for subsets of variables belonging to different modules, the following conditions are true:

\[
\bigcup_{i=1}^{2} Z_i = Z; \quad Z_i \cap Z_j = \emptyset; \quad |Z_2| \leq 1,
\]

where \( Z_i \) is the subset of variables, corresponding to the \( j \)-th module \((j = 1, 2)\).

Step 3. \( i = 1 \).

Step 4. If \( i = 1 \), then \( q \) is equal to the integer part of the ratio \(| Z | / 2\), otherwise \( q = |Z| - |Z_i|\).

Step 5. All the \( 2^q \) different states of the variables of the \( i \)-th module, as well as the components of the objective function and of constraints of problem (1) corresponding to each such state, are generated, and fixed in the RAM.

Step 6. \( i = i+1 \).

Step 7. If \( i > 2 \) then go to the step 8, otherwise go to the step 4.

Step 8. A new vector of variables is generated by new combination of in-RAM components of variables belonging to different modules. If there is no such a combination, then go to step 12.

Step 9. The new value of \( F \) is obtained by substituting the vector of variables obtained at the previous step into system (1).

Step 10. If \( F \) is "better" than \( R \) and all constraints of system (1) are satisfied, then go to step 11, otherwise, go to step 6.

Step 11. \( R := F \), the new vector of variables obtained at the 8-th step of the last iteration is stored, go to step 6.

Step 12. The algorithm is completed. The values of \( R \) and of the vector of variables stored in memory at the last call of the 11-th step are problem (2) optimal solution.

Example 1. Using the above Algorithm 1, solve the knapsack problem [14] of the form:

\[
\begin{align*}
7z_1 + 2z_2 + 4z_3 + 5z_4 & \rightarrow \text{max}; \\
2z_1 + 4z_2 + 8z_3 + 3z_4 & \leq 12; \\
\square: & \ z_i = 1, 0.
\end{align*}
\]

Determine the effectiveness of the proposed approach in relation to problem (17), provided that \( m = 2 \).

1) \( R = - \infty \).

2) Distribution of variables into two modules satisfying system (10) - (12): \( Z_1 = \{z_1; z_2\}; Z_2 = \{z_3; z_4\} \).

| z1 | z2 | \( \Delta F_1 \) | \( \Delta b_1 \) | z3 | z4 | \( \Delta F_2 \) | \( \Delta b_2 \) |
|----|----|----------------|----------------|----|----|----------------|----------------|
| 0  | 0  | 0              | 0              | 0  | 0  | 0              | 0              |
| 0  | 1  | 2              | 4              | 0  | 0  | 1              | 5              |
| 1  | 0  | 7              | 2              | 1  | 0  | 4              | 8              |
| 1  | 1  | 9              | 6              | 1  | 1  | 9              | 11             |

3) Generation of all states of the vectors of variables of each module and calculation of the corresponding components of the objective function \( \Delta F_i \) and of the left side of the constraint of the system (17) \( \Delta b_i \) are presented in the Table 1 above.
4) Analysis of all the pairwise combinations of vectors of variables of both modules (repeating 16 times steps 8-11 of Algorithm 1) allows us to obtain a globally optimal problem (13) solution: \( R = F_{\text{max}} = 14, Z_{\text{opt}} = \{1,1,0,1\} \).

5) The effectiveness of the proposed approach in solving problem (6) is determined by the gain in running time of Algorithm 1 as compared to the traditional realization of enumeration method solving the same problem. In the latter case, it can be shown that the running time of the search for a knapsack problem globally optimal solution is determined by the expression:

\[
T_1 = 2 \cdot (n-1) \cdot 2^n \cdot t_0, \tag{14}
\]

where \( t_0 \) is the same, as used above in (2).

A similar approach to estimating the search time for the same problem solution by modular enumeration allows us to determine this time as follows:

\[
T_2 = t_0 \cdot [n \cdot 2^{0.5n} + 2^{n+1}], \tag{15}
\]

The gain in the running time with the Algorithm 1 solving problem (6) is equal to \( \eta \):

\[
\eta = T_1 / T_2 \approx 1.33(3). \tag{16}
\]

The experimental results presented below in Section 4 confirm the assessment of the Algorithm 1 efficiency given in (5).

Unlike Algorithm 1, Algorithm 2, being also a modular enumeration one, is organized in such a way that not only the number of arithmetic operations at each iteration, but also the number of iterations is minimized during the enumeration process. This is achieved by changing the order of generation of the vectors of variables at step 8: during each iteration at this step a new vector of variables is generated, which consists of two parts corresponding to the best values of goal function components in each module. This strategy allows us not to analyze vectors of variables, which correspond to a priori "bad" values of the objective function, thus giving us an opportunity to get a gain in time exceeding that which is determined by (5).

Since the first seven steps of Algorithms 1 and 2 are the same, the description of Algorithm 2 below begins with the eighth step.

Algorithm 2

Step 8. All vectors of variables of the first module are considered as unlabeled.

Step 9. Among the unlabeled variables’ vectors of the first module, the one that corresponds to the best value of the components of objective function belonging to this module, is selected. If all vectors of variables belonging to this module are labeled, then go to step 17.

Step 10. All vectors of variables of the second module are considered as unlabeled.

Step 11. Among the second module unlabeled vectors of variables, the one that corresponds to the best value of the components of objective function belonging to this module, is selected. If all vectors of variables belonging to this module are already labeled, then go to step 9.

Step 12. Generation of a new vector of variables, the values of which were sequentially obtained at steps 9 and 11 of the last iteration.

Step 13. The vectors of the variables selected in steps 9 and 11 of the last iteration are labeled.

Step 14. If the value of the objective function, calculated with the values of the variables determined at step 12 of the last iteration, is "not better" than the previously found and stored in RAM value, then go to step 9, otherwise go to step 15.

Step 15. If the vector of variables created at step 12 of the last iteration satisfies all the conditions of problem (1), then go to step 16, otherwise, go to step 11.

Step 16. Replacing the value of the objective function stored in RAM with a new one corresponding to the vector of variables generated at step 12 of the last iteration and transit to step 9.

Step 17. The algorithm is completed. The best value of the objective function and corresponding vector of variables are stored in RAM.
It is easy to verify that, due to steps 9 and 11, the duration of each iteration of Algorithm 2 exceeds the similar parameter of Algorithm 1, however, due to steps 14 - 16, there is a good chance that the number of these iterations will decrease, and these possibility increase with the number of variables. The effectiveness of the above algorithm 2 is demonstrated below in two ways:

a) as applied to the solution of problem (13);

b) as shown by the statistical research presented in the next section.

Example 2. Using the above Algorithm 2, we solve the knapsack problem (13).

The first seven steps of solving problem (13) by algorithms 1 and 2 coincide and lead to the construction of Table 1 (above), that is why they are not presented here. The sequence of problem (13) vectors of variables generated and analysed by Algorithm 2 is shown below in Table 2. Here F denotes the current value of the objective function, G denotes the current value of the left side of the inequality of problem (13), B is the right side of this inequality, and R is the maximum allowable value of the objective function.

**Table 2. Vectors of variables analysed by Algorithm 2**

| № | x₁ | x₂ | x₃ | x₄ | F  | G  | R  | Remarks |
|---|----|----|----|----|----|----|----|---------|
| 1 | 1  | 1  | 1  | 1  | 18 | 17 | -∞ | G>B     |
| 2 | 1  | 1  | 0  | 1  | 14 | 9  | 14 | G<B     |
| 3 | 1  | 0  | 1  | 1  | 16 | 13 | 14 | G>B     |
| 4 | 1  | 0  | 0  | 1  | 12 | 5  | 14 | F<R     |
| 5 | 0  | 1  | 1  | 1  | 11 | 9  | 14 | F<R     |
| 6 | 0  | 0  | 1  | 1  | 9  | 11 | 14 | F<R     |

The obtained result coincides with the previously found with Algorithm 1, but the difference lies in the fact that Algorithm 2 needed to analyze only 6 plans, whereas Algorithm 1 needed to analyze 16 plans. The latter does not mean that the search time for a solution to problem (13) by Algorithm 2 is less than that with Algorithm 1: as noted above, the duration of iterations of Algorithm 2 exceeds the duration of iterations of Algorithm 1. A detailed experimental analysis of the efficiency of the modular enumeration methods described above in relation to the knapsack problem is presented below in Section 4.

The third modification of the modular enumeration differs from the algorithms above: that is before the actual enumeration begins, the components of both modules are arranged in the order of deterioration of "their" objective function components. This approach simplifies the procedure of enumeration and, at the same time, gives us chance to reduce its running time by ignoring a priori "bad" vectors of variables. As the first seven steps of Algorithms 1 and 3 coincide, the description of Algorithm 3 below begins with the eighth step.

**Algorithm 3**

Step 8. All the components of both modules are arranged in the order of deterioration of their components of goal function.

Step 9. i=1

Step 10. j=1.

Step 11. A new, not previously analyzed vector of variables is obtained by combining its i-th component of the first module and the j-th component of the second module.

Step 12. Calculation of a new value of the objective function F, corresponding to the vector obtained at the previous step.

Step 13. If F is "better" than R, then go to the next step, otherwise go to step 16.

Step 14. If the constraints are met, then go to step 15, otherwise go to step 19.

Step 15. R:= F, the new vector of variables obtained at the 11-th step of the last iteration is stored in RAM, go to the next step.
Step 16. If the components of problem (1) full vector of variables corresponding to the second module have values that coincide with the first ordering vector of the elements of this module obtained at step 8, then go to step 21.

Step 17. i = i + 1.

Step 18. If i > 2 |Z1|, then go to step 21, otherwise go to step 10.

Step 19. j = j + 1.

Step 20. If j < 2 |Z2|, then go to step 11, otherwise go to step 16.

Step 21. The algorithm is over. The best value of the objective function R and corresponding vector of variables are stored in RAM.

Example 3. Using the above Algorithm 3, solve the knapsack problem (13). As in the previous case the first seven steps of solving problem (13) by algorithms 1 and 3 coincide and lead to the construction of Table 1 (above). During step 8 of Algorithm 3 Table 1 is transformed into Table 3 below:

Table 3. All the components of both modules are arranged in the order of deterioration of their goal function components.

|   | z₁ | z₂ | ΔF₁ | Δb₁ | z₃ | z₄ | ΔF₂ | Δb₂ |
|---|----|----|-----|-----|----|----|-----|-----|
| 1 | 1  | 1  | 6   | 9   | 1  | 1  | 9   | 11  |
| 2 | 0  | 1  | 2   | 4   | 0  | 1  | 5   | 3   |
| 3 | 0  | 0  | 0   | 0   | 0  | 0  | 0   | 0   |

The sequence of problem (13) vectors of variables generated and analysed by Algorithm 3 is shown below in Table 4. In this case, the same designations as in Table 2 are used.

Table 4. Vectors of variables analysed by Algorithm 3

| №  | x₁ | x₂ | x₃ | x₄ | F  | G  | R  | Remarks |
|----|----|----|----|----|----|----|----|---------|
| 1  | 1  | 1  | 1  | 1  | 18 | 17 | -∞ | G>B     |
| 2  | 1  | 1  | 0  | 1  | 14 | 9  | 14 | G=B     |
| 3  | 1  | 0  | 1  | 1  | 16 | 13 | 14 | G>B     |
| 4  | 1  | 0  | 0  | 1  | 12 | 5  | 14 | F<R     |
| 5  | 0  | 1  | 1  | 1  | 11 | 15 | 14 | F<R     |

It is easy to verify that:

a) the volume of enumeration by Algorithm 3 was the smallest when solving (13) as compared to Algorithms 1 and 2;

b) this version of modular enumeration is close to the “meet in the middle” algorithm [16];

c) the results of solving (13) by all three algorithms above coincide.

4. Experimental Verification

The purpose of the experiments was to test the effectiveness of the above algorithms 1 - 3 in comparison with the traditional organization of exhaustive search in relation to the knapsack problem, the number of variables of which varied in the range from 3 to 20. Each algorithm’s criterion of efficiency in relation to knapsack problems of a fixed dimension resulted in an average gain in a problem-solving time. The behaviour of the algorithms was analysed in relation to two types of environments: benevolent and hostile. Benevolent environment below corresponds to the case, when in (1) aᵢ is close to the value ∑ bᵢ, whereas hostile environment is simulated by the opposite condition: aᵢ is close to the minᵢ{bᵢ}.
The computer with the following parameters was used during our experiments:
Processor Celeron N4100/J4125; RAM 8 Gb.; hard disk 1 Tb.
Operating system Windows 10 – 64.
The order of the experiments was determined by the following procedure:

Algorithm 4

Step 1. n = 3.
Step 2. Using Monte Carlo method we generated all the integer coefficients and constants of problem (1) for a given number of Boolean variables n in the range 1 - 10. Other parameters: d = 1, goal functions are maximized – generated are knapsack problems with the number of variables in the range 3 – 20.
Step 3. The problem obtained in the previous step is sequentially solved by software implementations of algorithms 1, 2, 3 and brute force. For each i-th algorithm (i=1, 2, 3) fixed are the corresponding running times T_i(n), T_2(n), T_3(n), whereas T(n) is the running time of the traditional organization of exhaustive search and the gains in running time \( \eta_1(n) = T(n)/T_1(n) \), \( \eta_2(n) = T(n)/T_2(n) \), \( \eta_3(n) = T(n)/T_3(n) \). Steps 2, 3 are repeated 10 times, after which the average values of the gain in time for each algorithm are recorded in the memory.
Step 4. n = n + 1.
Step 5. If n < 21, then go to step 2; otherwise, go to step 6.
Step 6. The algorithm is completed.

Experimental dependences of average values of the gain in time as function of number of variables n for each algorithm - \( \eta_1(n) \) (grey squares) for algorithm 1, \( \eta_2(n) \) (black oblique crosses) for algorithm 3 and \( \eta_3(n) \) (black triangles) for algorithm 2, are presented below in Figure 1 and Figure 2.

5. Conclusions

1. In contrast to the methods of implicit enumeration, such as B&B, dynamic programming and backtracking, described above approach allows one to create algorithms permitting a priori to evaluate their effectiveness regardless of the specific numerical values of the coefficients and constants of problem (1).
2. The efficiency of these algorithms depends on the environment in which they operate, but their common feature is the increase in efficiency with an increase in the number of variables of the problem being solved.
3. It is easy to make sure that in the case when the number of modules m is equal to two, the third modification of modular enumeration is close to the “meet in the middle” algorithm [16].
4. The gain in the running time presented by (5) is true only for the algorithm 1 above.
References

[1] D. Conte, B. Paternoster, L. Morandi, F. Mohammadi, 2019, Construction of Exponentially Fitted Explicit Peer Methods, International Journal of Circuits, Systems and Signal Processing, Volume 13, pp. 501-506.

[2] T. Atar, T. Mecabe, E. Erdogmus, B. Pulatsu, 2020, Dynamic Analysis of Semi-Circular Dry-Joint Masonry Arches: Small-Scale Experiment and Discrete Element Modeling, WSEAS Transactions on Computers, ISSN / E-ISSN: 1109-2750 / 2224-2872, Volume 19, Art. #19, pp. 137-142.

[3] K. Bibi, T. Feroze, 2020, Discrete Symmetry Transformations of Third Order Ordinary Differential Equations and Applications, WSEAS Transactions on Mathematics, ISSN / E-ISSN: 1109-2769 / 2224-2880, Volume 19, Art. #54, pp. 503-509

[4] J. S. Yeomans, 2019, Water Management using Population-based Simulation-Optimization, WSEAS Transactions on Information Science and Applications, ISSN / E-ISSN: 1970-0832 / 2224-3402, Volume 16, Art. #19, pp. 164-173.

[5] A. H. Land & A. G. Doig. 1960, An automatic method of solving discrete programming problems, Econometrica, Vol. 28, No. 3, pp. 497-520.

[6] C. A. Brown and P. W. Purdom Jr. 1982. An empirical comparison of backtracking algorithms. IEEE PAMI, pp. 309-315.

[7] F. Rossi, P. V. Beck, T. Walsh. 2006. Constraint satisfaction: an emerging paradigm. Handbook of Constraint Programming. Foundations of Artificial Intelligence. Amsterdam: Elsevier. p. 14. ISBN 978-0-444-52726-4.

[8] R. Bellman, 1954. The Theory of Dynamic Programming. The RAND Corporation.

[9] G. Desaulniers, J. Desrosiers, S. Spoorendonk. 2011. Cutting Planes for Branch-and-Price Algorithms. Published online 29 October 2011 in Wiley Online Library (wileyonlinelibrary.com). Wiley Periodicals, Inc.

[10] D. R. Morrison, 2014. New methods for branch-and-bound algorithms. Dissertation submitted for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign.

[11] V. O. Groppen, and A. A. Berko. 2019. Composite version of B&B algorithm: experimental verification of the efficiency. J. Phys.: Conf. Ser. 1278 012029, pp. 1-8. Available at: https://iopscience.iop.org/issue/1742-6596/1278/1

[12] V.O. Groppen. 2020. Composite Versions of Implicit Search Algorithms for Mobile Computing. Proceedings of the Future Technologies Conference (FTC) 2020, Volume 2, November 5–6, pp. 336 – 348.

[13] V.O. Groppen. 2021, Analysis of the Effectiveness of Composite Versions of Backtracking Algorithms. Conference proceedings, RusAutoConf 2020: Available: Advances in Automation II, Springer, pp 235-244.

[14] L. Caccetta, A. Kulanoot, 2001. Computational Aspects of Hard Knapsack Problems. Nonlinear Analysis, 47: 5547–5558. Available: doi:10.1016/s0362-546x(01)00658-7

[15] V.O. Groppen. 2021. A New Method Searching Globally Optimal Solutions to Discrete Optimization Problems. Proceedings of the Future Technologies Conference (FTC) 2021, Volume 3, pp 486 – 494.

[16] H. Ellis, S. Sartaj. 1974. Computing partitions with applications to the knapsack problem, Journal of the Association for Computing Machinery, 21 (2), pp. 277–292.