A new family of algorithms searching optimal solutions to discrete optimization problems

Vitaly O. Groppen
North-Caucasian Institute of Mining and Metallurgy (State technological university).
Nikolaev str. 44, Vladikavkaz, 362021,
Russia

Received: August 2, 2021. Revised: November 21, 2021. Accepted: December 7, 2021.
Published: December 31, 2021.

Abstract - A new family of algorithms searching globally optimal solutions to discrete programming problems is proposed. It is proved that the effectiveness of this method depends only on the number of variables of the problem being solved. The main difference from the methods of implicit enumeration is in the ability to a priori predict, solving by the one of proposed methods any extreme problem with Boolean variables, the number of iterations, the running time, and the amount of RAM used. The results of experimental verification of the effectiveness of various versions of modular enumeration procedures and, on their basis, recommendations for the use of these algorithms are given.

Keywords — discrete programming, globally optimal solution, modular enumeration, solution search time.

I. INTRODUCTION

The interest in discrete optimization problems can be explained by a large number of reducible applications. To obtain globally optimal solutions of these problems, as a rule, are used either brute force methods or implicit enumeration methods developed in the middle of the last century, such as dynamic programming, branch-and-bound (B&B) methods, backtracking [1]-[4] and their modifications [5]-[9]. Keeping in mind all the positive features of all these methods, nevertheless, it should be recognized that all these methods have several negative features:

- the impossibility to predict a priori neither the running time nor the gain in running time from the use of these procedures as compared to the brute force methods;
- operating in a hostile environment, i. e. in the case when B&B method, as well as backtracking, makes many mistakes when choosing the direction of the search in the branch tree, there may be cases when the number of iterations by these methods exceeds the number of different complete plans, thus running time of implicit enumeration methods is exceeding the brute force method running time;
- using dynamic programming the number of analyzed plans can exceed the number of different complete plans, when, during search, it is possible to cut off only a relatively few unpromising groups of vectors of variables.

At the same time, the brute force method makes it possible to predict a priori the running time for globally optimal solutions search to specific problems. That is why, the idea of applying and improving the method of complete enumeration of all combinations of values of variables in relation to discrete optimization problems remains relevant. This approach is also so attractive because it allows you to a priori predict the number of iterations and the running time. The main idea of the proposed below two algorithms is in the modular organization of complete enumeration permitting to shorten its running time thus increasing its performance due to the more intensive use of the computer's RAM. Thus, within the framework of the proposed approach, we pay with RAM for an increase in the computational speed.

This paper contains description, analysis and experimental verification of efficiency of the two new realizations of the brute force method including its modular organization related to searching a globally optimal solution to extreme problems with discrete variables. Examples and statistics below are based on knapsack problems solving [10].

II. BASIC PRINCIPLES 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) usage of the contents of the modules for generation of all complete plans, the corresponding values of the objective function and constraints;
b) their comparison and selection of the best values of the vector of variables.
It is easy to show that in the case of m modules, regardless of the number of variables of a knapsack problem being solved, only (m-1) operations of addition and two comparison operations at each iteration are performed.
This raises two questions:
1) What should be the strategy of variables distribution between modules?
2) What is the optimal number of modules for a specific task?
The answers are contained in the two theorems below:
Theorem 1. The optimal strategy corresponds to such a distribution of variables over modules that the numbers of different vectors of variables for any pair of modules coincide.
The proof of Theorem 1 is given in Appendix.
The multiplicity constraint for “n” and “m” is not strong due to the following theorem:
Theorem 2. The optimal number of modules “m” for problems with Boolean variables does not depend on the number of variables and constraints of problem to be solved and is equal to 2: $m_{opt} = 2$. (1)
The proof of Theorem 2 is given in Appendix.
These two theorems allow us the following step by step description of two Algorithms, guaranteeing minimal running time for problems with Boolean variables of the form [7]–[9]:

$$\begin{cases} F = \sum_{i}^{n} C_i z_i \rightarrow \max(min); \\ \sum_{i} b_{ij} z_i \le a_j; \;\; j = 1, 2, \dots d; \\ \forall i: z_i = 1,0, \end{cases} \qquad (2)$$

where $Z = \{z_1, z_2, ..., z_n\}$ is a vector of Boolean variables, whereas $\forall$i,j: C_i, b_{ij}, and a_j are integer constants.

III. MODULAR ENUMERATION ALGORITHMS

The main feature of Algorithm 1 presented below is such an organization of the full enumeration process, in which:
a) the conditions of Theorems 1 and 2 are satisfied;
b) the number of arithmetic operations is minimal.

Algorithm 1

Step 1. If problem (2) goal function F is maximized, then R= -∞, otherwise R=+∞.
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, subsets of variables in which do not intersect, in other words, for subsets of variables belonging to different modules, the following conditions are true:

$$\begin{cases} \bigcup_{j=1}^{j=2} Z_j = Z; & (3) \\ Z_1 \cap Z_2 = \emptyset; & (4) \\ |Z_2| - |Z_1| \le 1, & (5) \end{cases}$$

where Z_j 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_1|.

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 (2) 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 (2).
Step 10. If R is "better" than F, then go to step 6, otherwise, go to step 11.
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 over. 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. The number of analyzed vectors of variables is equal to (i - 2).

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

$$\begin{cases} 7z_1 + 2z_2 + 4z_3 + 5z_4 \rightarrow \max; \\ 2z_1 + 4z_2 + 8z_3 + 3z_4 \le 12; \\ \forall i: z_i = 1,0. \end{cases} \qquad (6)$$

Determine the effectiveness of the proposed approach in relation to problem (6), provided that m = 2.
1) R = - ∞.
2) Distribution of variables into two modules satisfying system (3) - (5): $Z_1 = \{z_1;z_2\}$; $Z_2 = \{z_3;z_4\}$.
3) Generation of all states of the vectors of variables of each module and calculation of the corresponding components of the objective function ΔF_i and of the left side of the constraint of the system (6) Δb_i are presented in the Table 1 bellow:

Table 1

m_1				m_2			
z_1	z_2	ΔF_1	Δb_1	z_3	z_4	ΔF_2	Δb_2
0	0	0	0	0	0	0	0
0	1	2	4	0	1	5	3
1	0	7	2	1	0	4	8
1	1	9	6	1	1	9	11

Components of modules m_1 and m_2

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 (6) solution: $R = F_{max} = 14$, $Z_{opt} = \{1,1,0,1\}$.

5) The effectiveness of the proposed approach in relation to problem (6) is determined by the gain in

running time of Algorithm 1 in comparison with 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:

$$T_1=2\cdot(n-1)\cdot 2^n\cdot t_0, \quad (7)$$

where t_0 is equal to the sum of the times of addition and multiplication of two integer numbers, whereas time of two numbers comparison is ignored. A similar approach to estimating the search time for the same problem solution by modular enumeration allows us to determine this time as follows:

$$T_2 = t_0\cdot[n\cdot 2^{0.5n} + 2^{n+1}]. \quad (8)$$

The gain in running time resulting algorithm 1 usage when solving problem (6) is equal to η:

$$\eta = T_1/T_2 \approx 1.33(3). \quad (9)$$

The gain in computation time from the use of Algorithm 1 as compared to exhaustive search is determined by the ratio of the right-hand sides (7) and (8):

$$\eta = T_1/ T_2 = (n-1)\cdot 2^{n+1}/[n\cdot 2^{0.5n} + 2^{n+1}]. \quad (10)$$

Since with growth of n the fraction of the first term in the denominator of the right-hand side of (10) in the sum of both terms decreases, the following equality is true:

$$\lim_{n\to\infty} \eta = n-1. \quad (11)$$

The experimental results presented below in Section 4 confirm the assessment of the efficiency of Algorithm 1 given in (11). It can be also shown that the amount of RAM used by Algorithm, solving problem (2) with n variables, is proportional to the value $V = n\cdot 2^{0.5n}$. (12)

Unlike Algorithm 1, Algorithm 2 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 is generated a new vector of variables, consisting of two parts corresponding to the best values of goal function components in each module. This strategy avoids the analysis of vectors of variables, which correspond to a priori "bad" values of the objective function, thus, giving the opportunity to get a gain in time exceeding that which is determined by (11).

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 vectors of variables 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 unlabeled vectors of variables of the second 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 already labeled, then transition 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 (2), 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 transition to step 9.
Step 17. The algorithm is over. The best value of the objective function is 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, thanks to steps 14 - 16, there are chances that the number of these iterations will decrease, and these chances 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 (6);
b) statistical research presented in the next section.

Example 2. Using the above Algorithm 2, solve the knapsack problem (6).
The first three steps of solving problem (6) by algorithms 1 and 2 coincide and lead to the construction of Table 1 (above), that is why they are not presented below. The sequence of problem (6) vectors of variables generated and analyzed 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 (6), B is the right side of this inequality, and R is the maximum allowable value of the objective function.

Table 2

#	x_1	x_2	x_3	x_4	F	G	Remarks
1	1	1	1	1	18	17	G>B
2	1	1	0	1	14	9	G<B, R=14
3	1	0	1	1	16	13	G>B
4	1	0	0	1	12	5	F<R
5	0	1	1	1	11	15	F<R
6	0	0	1	1	9	11	F<R

The sequence of vectors of variables analyzed by Algorithm 2

The result coincides with the previously found by Algorithm 1, but the difference lies in the fact that Algorithm 2 needed to analyze only 6 plans for this, whereas Algorithm 1 needed to analyze 16 plans. The latter does not mean that the search time for a solution to problem (6) by Algorithm 2 is less than by 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.

IV. Experimental Verification of Modular Enumeration Effectiveness

During experiments was used computer with the following parameters: 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 3

Step 1. $n = 3$.
Step 2. By the use of Monte Carlo method generated are all the integer coefficients and constants of problem (2) for a given number of Boolean variables n in the range 1 - 10. The 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 and 2 and brute force. For each algorithm fixed are the running times $T_1(n)$, $T_2(n)$, $T(n)$ and the gains in running time $\eta_1(n)= T(n)/T_1(n)$, $\eta_2(n)= T(n)/T_2(n)$. 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 end of the algorithm.

Experimental dependences of average values of the gain in running time in comparison with the traditional realization of enumeration method for each algorithm above - $\eta_1(n)$ (grey squares) for algorithm 1, $\eta_2(n)$ (black oblique crosses) for algorithm 2 and corresponding to (11) line $\eta(n)$ (black triangles), are presented below in Fig. 1.

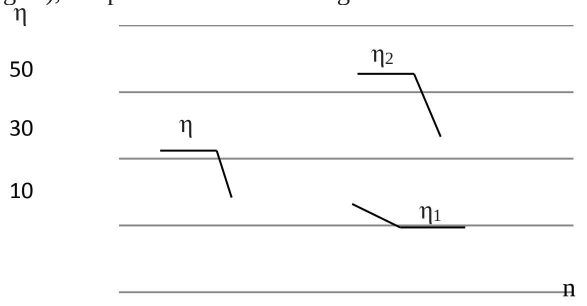

Fig. 1. Algorithms 1 and 2 analytic and experimental gains in running time.

It is easy to verify that the experimental data behave themselves as predicted in Section 2: with an increase in the number of variables, the value of the gain in time for solving the knapsack problem by Algorithm 1 in comparison with brute force search approaches the value n. In addition, it has been experimentally confirmed that when solving problems with a relatively small number of variables, Algorithm 1 is more efficient than Algorithm 2; however, with an increase in the number of variables, the higher efficiency of the possibility of cutting off subsets of "bad" vectors of variables contained in Algorithm 2 becomes obvious.

V. Conclusions

In contrast to the methods of implicit enumeration, such as B&B, dynamic programming and backtracking, proposed above algorithm 1 to solving problems of type (2) allows a priori determination of:
a) the computational time;
b) the amount of used RAM;
c) gain in computation time as compared to exhaustive search, regardless of the specific numerical values of the coefficients and constants of problem (2). The efficiency of this approach increases with the growth of this problem number of variables n and for sufficiently large n this dependence is close to the linear one, corresponding to (11). As for Algorithm 2, it has been experimentally shown that for sufficiently large values of n, its efficiency exceeds the efficiency of Algorithm 1, allowing one to use (11) as a lower bound of the gain in this algorithm's running time as compared with the brute force method.
Further development of modular enumeration can be associated with the two directions: extension of this approach to the case of non-Boolean variables; development of such modular enumeration algorithms that allow adjusting the amount of used RAM.

References

[1] A.H. Land, A. G. Doig. An automatic method of solving discrete programming problems, Econometrica, Vol. 28, No. 3, 1960, pp. 497-520.
[2] C. A. Brown and P. W. Purdom Jr. An empirical comparison of backtracking algorithms. *IEEE PAMI*, 1982, pp. 309-315.
[3] F. Rossi, P.V. Beek, T. Walsh, "Constraint Satisfaction: An Emerging Paradigm". Handbook of Constraint Programming. Foundations of Artificial Intelligence. Amsterdam*:* Elsevier. *p. 14*. ISBN 978-0-444-52726-4, 2006.
[4] R. Bellman. The Theory of Dynamic Programming. *The* RAND Corporation, 1954.
[5] G. Desaulniers, J. Desrosiers, S. Spoorendonk. Cutting Planes for Branch-and-Price Algorithms. Published online 29 October 2011 in Wiley Online Library (wileyonlinelibrary.com). © Wiley Periodicals Inc., 2011.
[6] D. R. Morrison 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, 2014.
[7] V. O. Groppen, and A.A. Berko Composite version of B&B algorithm: experimental verification of the efficiency. J. Phys.: Conf. Ser. 1278 012029, 2019, pp. 1-8. https://iopscience.iop.org/issue/1742-6596/1278/1
[8] V.O. Groppen. Composite Versions of Implicit Search Algorithms for Mobile Computing. Proceedings of the Future Technologies Conference (FTC) 2020, Volume 2, November 5–6, 2020, pp. 336 – 348.

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

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

Appendix

Proof of Theorem 1.
The running time T_1 corresponding to the search by algorithm, satisfying condition (8), for a globally optimal solution to problem (1) is the sum of two times: T_{11} and T_{12}: $T_1 = T_{11} + T_{12}$, where T_{11} - time spent by algorithm 1 on steps 3 - 6, T_{12} - time spent by this algorithm on steps 7 - 11. Taking into account (8), we get (two numbers comparison time is ignored):

$$T_{11} = t_0 \cdot \cancel{m} \cdot \frac{n}{\cancel{m}} \cdot 2^{n/m} = t_0 \cdot n \cdot 2^{n/m}; \qquad (A.1)$$

$$T_{12} = m \cdot 2^n \cdot t_0, \qquad (A.2)$$

where t_0 is the same, as used above in (7).
We obtain a new, non-uniform distribution of variables between the modules, for which in the uniform distribution we remove one variable from the module k and transfer it to the module q. The search time for problem (1) solution by Algorithm 1 in this case is denoted as T_2: $T_2 = t_0 \cdot [\frac{n}{m} \cdot (m-2) \cdot 2^{n/m} + (\frac{n}{m} - 1) \cdot 2^{n/m-1} + (\frac{n}{m} + 1) \cdot 2^{n/m+1} + m \cdot 2^n]$. (A.3)
The difference $T_2 - T_1$ is designated bellow as ΔT:

$$\Delta T = t_0 \cdot [\frac{n}{m} \cdot (m-2) \cdot 2^{n/m} + (\frac{n}{m} - 1) \cdot 2^{n/m-1} + (\frac{n}{m} + 1) \cdot 2^{n/m+1} + m \cdot 2^n - n \cdot 2^{n/m} - m \cdot 2^n].$$

After expanding the parentheses and transformations, we get:

$$\Delta T = 0.5 \cdot 2^{n/m} \cdot t_0 \cdot [\frac{n}{m} + 3]. \qquad (A.4)$$

Since n and m are non-negative integers, ΔT is also non-negative. It follows that $T_1 < T_2$. The theorem is proved.

Proof of Theorem 2.
Below are compared three search times for problem (1) globally optimal solution: T_1 - brute force time; T_2 is the time spent on finding globally optimal solution to the same problem by Algorithm 1, provided that $m = 2$; T_3 is the Algorithm 1 running time , provided that $m = 2+i$, where $i > 0$, integer. Keeping in mind, that in any case ratio n/m is integer and not less than 2, corresponding analytical expressions are presented below:

$$T_1 = n \cdot 2^n \cdot t_0; \qquad (A.5)$$

$$T_2 = [n \cdot 2^{n/2} + 2^{n+1}] \cdot t_0; \qquad (A.6)$$

$$T_3 = [n \cdot 2^{n/(2+i)} + (2+i) \cdot 2^n] \cdot t_0. \qquad (A.7)$$

The difference $T_i - T_j$ is denoted below as Δ_{ij}:

$$\Delta_{12} = [n \cdot 2^n - n \cdot 2^{n/2} - 2^{n+1}] \cdot t_0; \qquad (A.8)$$

$$\Delta_{32} = [n \cdot 2^{n/(2+i)} + 2^{n+1} + i \cdot 2^n - n \cdot 2^{n/2} - 2^{n+1}] \cdot t_0. \qquad (A.9)$$

After conversions, (A.9) can be presented to the form:

$$\Delta_{32} = [n \cdot 2^{n/(2+i)} + i \cdot 2^n - n \cdot 2^{n/2}] \cdot t_0. \qquad (A.10)$$

Taking into account that for all integer and even n it is true:

$$2^{n/2} \geq n, \qquad (A.11)$$

it is easy to prove the inequality:

$$n \cdot 2^n - n \cdot 2^{n/2} - 2^{n+1} > n \cdot 2^n - 2^n - 2^{n+1}. \qquad (A.12)$$

But the right-hand side of (A.12) for the cases, when $n > 4$, is non-negative, resulting in $\Delta_{12} > 0$ that is, $T_1 > T_2$. (A.13)
Taking into account (A.11), true is the following inequality:

$$n \cdot 2^{n/(2+i)} + i \cdot 2^n - n \cdot 2^{n/2} > n \cdot 2^{n/(2+i)} + i \cdot 2^n - 2^n. \qquad (A.14)$$

But as “i” in (A.7) is integer and non-negative, the right-hand side of (A.14) is non-negative too:

$$n \cdot 2^{n/(2+i)} + i \cdot 2^n - 2^n = n \cdot 2^{n/(2+i)} + (i-1) \cdot 2^n - 2^n > 0, \qquad (A.15)$$

thus resulting in $\Delta_{32} > 0$ that is, $T_3 > T_2$. (A.16)
The simultaneous validity of inequalities (A.13) and (A.16) proves the validity of (1). Theorem 2 is proved.