CYBERNETICS and COMPUTER TECHNOLOGIES

UDC 519.168 DOI:10.34229/2707-451X.22.4.2

O. TROFYMCHUK, V. VASYANIN, V. SOKOLOV,
A. CHIKRII, L. USHAKOVA

ON THE USE OF GRAY CODES FOR SOLVING 0-1 COMBINATORIAL PROBLEMS OF OPTIMIZATION IN ENVIRONMENTAL AND ECONOMIC SYSTEMS

The article provides useful information for developers of algorithms and programs on the use of Gray codes for solving combinatorial problems with pseudo Boolean functions. As an example of the effectiveness of the use of these codes, the solution on two combinatorial problems with Boolean variables with a full search of the solutions is considered. The results of an experimental study are presented, which show that Gray codes can be practically applied in branching schemes, for example, in the branch and bound method, when the number of variables in the branch nodes of the decision algorithm does not exceed 35.

Keywords: *Gray codes, combinatorial optimization problems, problem solving time.*

Introduction. Environmental and economic problems are an example of a complex of tasks, the solution of which is associated with high costs, and the decisions themselves have a long-term large-term impact on the fate of large social groups. Any ecological and economic system is a complex, weakly deterministic and evolving research object. The choice of strategies for making environmental and economic decisions in such large-scale systems is associated with the analysis of a huge number of factors, relations and interests. This leads to the need of use a modern means of analysis, forecasting and mathematical modeling of the behavior of complex systems based on the information and computer technologies and decision support systems.

The foundations of mathematical modeling of complex ecosystems were developed by such scientists as V. Volterra, B.G. Zaslavsky, A.N. Kolmogorov, G.I. Marchuk, Yu. Odum, R.A. Poluektov, Yu.M. Svirezhev. A great contribution to the study of environmental and economic problems of environmental management was contributed by such scientists as O.F. Balatsky, K.G. Gofman, V.I. Gurman, A.V. Lotov, V.V. Leontiev, N.N. Olenev, A.A. Petrov, I.G. Pospelov, R.L. Rayatskas, G.A. Ugolnitsky, and others. Among the Ukrainian scientists it should be noted the works of A.A. Bakaev, A.P. Velikiy, V.M. Heyets, V.M. Glushkov, V.S. Grigorkiv, Yu.M. Ermolyev, S.I. Doroguntsov, A.G. Ivakhnenko, V.S. Mikhalevich, B.N. Pshenichnyi, N.Z. Shor.

Despite the deep scientific developments, the processes of environmental and economic cooperation require further study in order to develop new and improving the already existing methods for solving socio-economic problems and conservation of natural resource potential. The results of mathematical modeling of the scheme of an ecologically balanced economy of individual regions, and, in particular, their dynamic optimal development, can serve as a basis for making management decisions aimed at improving the efficiency of environmental and economic systems.

The tasks of the Boolean programming are played an important role in the study of these systems, and are the class of problems of discrete optimization, which are widely used at solving the problems of making schedules and destination, investment planning and placement of enterprises and the nodes of communication networks, unification and standardization and etc. As a target function in these problems use either the amount of total costs for the creation and operation of the system, or the total system efficiency, i.e. the amount of work performed. Numerous monographs [1–6] and articles (see for example [7–12]) are devoted to studying the problems of Boolean programming. Despite the simplicity of the wording of such tasks, most of them belong to the class of NP-hard problems [13]. Among the most popular methods for solving a problems of Boolean programming it is possible to allocate: exact methods - branch and bound method and dynamic programming and their combination, approximate algorithms with a guaranteed deviation, evolutionary (Genetic Algorithms, Ant Colonies Optimization) and metaheuristic algorithms (Simulated Annealing, Tabu Search, Greedy Randomized Adaptive Search Procedure - GRASP, Variable Neighborhood Search and etc.). In practice, the solution of these tasks requires large computational costs, which substantiates the feasibility of applying various ways to reduce the dimension of the problem, for example, using to reduction of source optimization problem to a problem of optimize the corresponding pseudo-Boolean function. Recall that the real functions defined on the set of Boolean variables, by analogy with the Boolean functions, are called pseudo-Boolean [14–17]. Accordingly, the problems of optimization of pseudo-Boolean functions are called the problems of pseudo-Boolean optimization. Any pseudo--Boolean function can be the only way presented as a multi-polynom type

$$F(x_1,...,x_n) = c_0 + \sum_{k=1}^{m} c_k \prod_{i \in A_k} x_i ,$$

where $c_0, c_1, ..., c_m$ – real coefficients, $A_1, A_2, ..., A_m$ – non-empty subsets $N \in \{1,2,...,n\}$ [16]. In addition, any pseudo-Boolean function can be represented as

$$F(x_1,...,x_n) = b_0 + \sum_{k=1}^{m} b_k \left(\prod_{i \in A_k} x_i \prod_{j \in B_k} \bar{x}_j\right),$$

where $b_0, b_1, ..., b_m$ – real coefficients, $\bar{x}_j = 1 - x_j$, $j = \overline{1,n}$. If $b_k \geq 0$, $k = \overline{1,m}$, then the second expression is the posi-form of the function F. Any pseudo-Boolean function can be recorded as the posi-form. The formal statement of the problem of conditional pseudo-Boolean optimization is as follows: $F(X) \to extr$, where $F : \Omega \to R^1$, and $\Omega \subset B_2^n$ – some sub-area of the space of Boolean variables, determined by the specified system of restrictions.

Many problems in environmental and economic systems lead to the need to solve conventional optimization problems with Boolean variables.

Another important feature in solving combinatorial problems with Boolean variables is to use Gray codes and parallel calculations to reduce the time of recalculation of the objective functions and restrictions in various schemes of branching the decisive algorithm (for example, the method of branches, borders and cutting, dynamic programming, etc.).

The paper discusses the use of binary-reflected Gray codes to solve combinatorial problems with pseudo-Boolean functions (polynomials from Boolean variables). The recursive G. Ehrlich algorithm is given for generate a sequence of strings n - discharge Gray codes, in which each next string differs from the previous one by one discharge (bit). On the examples of the solution of the 0-1 Knapsack Problems [18] and the Problems of Choosing the Capacity of Arcs with Constraint on Flow Delay Time [19], it is shown how these codes can be used to effectively calculate the values of the target function and restrictions.

The purpose of the article is to show the developers of algorithms and programs how to apply Gray codes in various branching schemes of the decision algorithm, for example, in the branch and bound method, when the number of binary (Boolean) variables at the nodes of the tree is small (less than 35).

The research methodology is based on a computational experiment for solving the above problems with the proposed algorithm of exhaustive search of the solution with partial and full recalculation of the values of objective function and constraint of the problem. During the experiment, the accuracy of solving the 0-1 Knapsack Problem by a "greedy" heuristic algorithm with time complexity $O(n^2)$ was also checked.

1. 0-1 Knapsack Problem and Problem of Choosing the Capacity of Arcs

The mathematical formulation of the first problem is as follows. A set of n items is set, for each of which the cost $c_i \in Z^+$ and weight $a_i \in Z^+$, $i = \overline{1,n}$ are known. It is required to load the knapsack with the items so that the total profit of the selected items is maximized and the total weight does not exceed $W \in Z^+$:

$$\max \sum_{i=1}^{n} c_i x_i \tag{1}$$

s.t.

$$\sum_{i=1}^{n} a_i x_i \le W \tag{2}$$

$$x_i \in \{0,1\}, i = \overline{1,n}. \tag{3}$$

It is assumed that $a_i \le W$, $i = \overline{1,n}$ and $\sum_{i=1}^{n} a_i > W$.

The second problem is to choose an arcs capacity from a given set of discrete integer values when limited to the maximum flow delay time, which is relevant for the distribution flows in multicommodity communication networks. In this problem delays of flows t_{kl} on arcs are defined as $t_{kl} = f_{kl} / (w_{kl} - f_{kl})$, $kl \in E$, and the constraint on the delay time of flows t_{av} in a network has the following form $t_{av} = 1/U_\Sigma \sum_{kl \in E} f_{kl} / (w_{kl} - f_{kl}) \le T_{\max}$. Here $f_{kl} \in Z^+$ – fixed arc flow value for $kl \in E$, E – set of arcs of network, $w_{kl} \in Z^+$ – bandwidth capacity of arc $kl \in E$, $T_{\max}$ – the maximum of flows delay time in network, $U_\Sigma = \sum_{ij \in S} u_{ij}$ – total flow in network, $u_{ij} \in Z^+$ – value of the flow from a node i to a node j, S – set of pairs of indexes corresponding nodes in the network.

When approaching the magnitude of the flow on the arcs to their carrying capacity, the delay increases and, therefore, network congestion can occur. The essence of the problem is for fixed flows it is necessary to choose the throughput capacities of arcs from a given set of integers so that the constraint on the delay time of flows is fulfilled and the minimum of some objective function is achieved. This problem also arises in transport networks when distributing flows according to the criterion of the minimum cost of the network and a given restriction on the delay time of flows [20–22]. By controlling the parameter $T_{\max}$ for maximum delay, the data network administrator or the transport network manager can provide the required reserve for the bandwidth capacity of the communication channels or the carrying capacity of vehicles at predicted fluctuations of values of flows on a given time of intervals. A decrease in the parameter $T_{\max}$ (increase in the reserve) leads to a rise in the cost of the network, but reduces the probability of redistribution of flows

and technical re-equipment of communication channels or fleet of vehicles at increasing flows and a threat of emergence overloads in the network. An increase in the parameter $T_{\max}$ makes it possible to reduce the capacity of communication channels or the carrying capacity of vehicles and the cost of the network, but increases the risk of redistributing flows and upgrading the network.

We consider a direct connected network $G(N,E)$ with a set of nodes N, $n=\left|N\right|$ and a set of arcs E, $e=\left|E\right|$. In network for each direct arc kl, ($k<l$) exist back arc lk, ($l>k$). An arc represents a switched communication line in a data network or a vehicle route, the final nodes of which coincide with the initial and final node of the arc. The network may contain loops and parallel arcs, since cyclic and repeating communication lines and communication lines with the same final nodes are allowed. An integer flow matrix is given on the network $U=\left\| u_{ij} \right\|_{n\times n}$. Let w_{kl}, $kl\in E$ – sought-for a bandwidth capacity of arcs of the network in transport blocks, $w_{kl}\in\{w_1,w_2,...,w_\alpha\}$, w_i, $i=\overline{1,\alpha}$ – ascending positive integers; $d_{kl}\in R^+$, $kl\in E$ – arcs lengths; $C_{kl}(w_{kl},d_{kl})\in R^+$, $kl\in E$ – discrete values cost of arcs, such that $C_{kl}(w_i,d_{kl})\le C_{kl}(w_{i+1},d_{kl})$, $i=\overline{1,\alpha-1}$; $f_{kl}=\sum\limits_{ij\in S} u_{ij}^{kl}$, $kl\in E$ – fixed total flows in transport blocks, a flowing along the arcs of the network, where u_{ij}^{kl} – is the flow of transport blocks from i to j, which passes along arc kl.

It is required to find the minimum value of the network cost function

$$\min_{w_{kl}} \sum_{kl\in E} C_{kl}(w_{kl},d_{kl}),\ w_{kl}\in\{w_1,w_2,...,w_\alpha\} \tag{4}$$

s.t.

$$\frac{1}{U_\Sigma}\sum_{kl\in E}\frac{f_{kl}}{w_{kl}-f_{kl}}\le T_{\max},\ w_{kl}>f_{kl},\ \forall kl\in E. \tag{5}$$

Note that problem (4), (5) can be represented as a knapsack problem with Boolean variables and multichoice (0-1 Multiple-choice Knapsack Problem, 0-1 MCKP). Let $c_{ij}\in R^+$ – discrete values cost of arcs i with capacity $w_{ij}\in\{w_1,w_2,...,w_\alpha\}\in Z^+$ and length d_i, $j=\overline{1,\alpha}$, $i=\overline{1,e}$; $t_{ij}=f_i/(w_{ij}-f_i)$, $w_{ij}>f_i$, $j=\overline{1,\alpha}$, $i=\overline{1,e}$ – delays of flows on arcs; f_i – flow on the arc i, $i=\overline{1,e}$. Suppose that $x_{ij}=1$, if for the arc i the capacity w_{ij} is selected, $j=\overline{1,\alpha}$, $i=\overline{1,e}$, and $x_{ij}=0$ otherwise. We require to find

$$\min\sum_{i=1}^{e}\sum_{j=1}^{\alpha}c_{ij}x_{ij} \tag{6}$$

s.t.

$$\frac{1}{U_\Sigma}\sum_{i=1}^{e}\sum_{j=1}^{\alpha}t_{ij}x_{ij}\le T_{\max}, \tag{7}$$

$$\sum_{j=1}^{\alpha}x_{ij}=1,\ i=\overline{1,e}, \tag{8}$$

$$x_{ij}\in\{0,1\}. \tag{9}$$

Here, the required throughputs w_{ij} correspond to the optimal solution x_{ij}^* to the problem (6)–(9).

It is easy to see that any individual problem formulated in the form of (4), (5) can be transformed in time $O(e\alpha)$ into the corresponding instance of problem (6)–(9). To do this, it is necessary to construct two matrices of size $e \times \alpha$, whose rows correspond to arcs, the columns – to a set of discrete capacities, and the cost of arcs c_{ij} and delays on arcs t_{ij} are taken as matrix elements. The converse is also true.

In [19] it has been proven that the optimization problem formulated in the form (4), (5) is NP-hard, and for its solution two approximate algorithms were proposed on the basis of the approximation of discrete cost functions by linear ones, and on the method of sequential analysis of options. The first algorithm uses the Lagrange multiplier method, which allows one to analytically solve a relaxed problem and obtain an exact continuous solution. The second algorithm enumerates the solutions, narrowing the range of feasible solutions at each iteration, and can be used for any monotonically non-decreasing cost of arcs with an increase in their throughput. It can be applied both to the initial statement of the problem, and to the statement in the form (6)–(9). It was shown that both algorithms at the final stage of work narrow the area of permissible solutions up to two values $w_{ij} \in \{w_j, w_{j+1}\}$, $j = \overline{k, \alpha - 1}$, $i = \overline{1, e}$, and with complete full-search 2^e variants allow us to obtain an exact solution.

We write the abbreviated problem in the form:

$$\min \sum_{i=1}^{e} \sum_{j=1}^{2} c_{ij} x_{ij} \tag{10}$$

s.t.

$$\frac{1}{U_\Sigma} \sum_{i=1}^{e} \sum_{j=1}^{2} t_{ij} x_{ij} \le T_{\max}, \tag{11}$$

$$\sum_{j=1}^{2} x_{ij} = 1, \; i = \overline{1, e}, \tag{12}$$

$$x_{ij} \in \{0,1\} \tag{13}$$

were c_{ij}, $j = \overline{1,2}$, $i = \overline{1,e}$ – discrete cost of arcs; $t_{ij} = f_i / (w_{ij} - f_i)$, $j = \overline{1,2}$, $i = \overline{1,e}$ – delays of flows on arcs; f_i – flow on the arc i, $i = \overline{1,e}$.

Problem (10)–(13) can be converted to 0-1 Knapsack Problem:

$$\min \sum_{i=1}^{e} (c_{i1} + \Delta c_i x_i) \tag{14}$$

s.t.

$$\frac{1}{U_\Sigma} \sum_{i=1}^{e} (t_{i1} - \Delta t_i x_i) \le T_{\max}, \tag{15}$$

$$x_i \in \{0,1\}, \; i = \overline{1, e}, \tag{16}$$

were $\Delta c_i = c_{i2} - c_{i1}$, $t_{i1} = f_i / (w_{i1} - f_i)$, $\Delta t_i = f_i / (w_{i1} - f_i) - f_i / (w_{i2} - f_i)$, $i = \overline{1,e}$, $\frac{1}{U_\Sigma} \sum_{i=1}^{e} t_{i1} > T_{\max}$, $\frac{1}{U_\Sigma} \sum_{i=1}^{e} t_{i2} \le T_{\max}$.

All formulated problem are NP-hard, i.e., in general, there are no exact polynomial algorithms to solve them [13]. In books [1, 5], you can find a description of the three exact pseudo-polynomial algorithms for

solving problems (1)–(3) and (14)–(16) based on the methods of branches and bounds and dynamic programming with using Lagrangian and LP relaxation. The programs listing of these algorithms (Expknap, Minknap and Combo) on C ++ are shown on the D. Pisinger page on the Internet (www.diku.dk/~pisinger/). In the same books, there are also completely polynomial approximate schemes for solving 0-1 KP problem (FPTAS – Fully Polynomial Time Approximation Scheme). This means that for them there are algorithms that, polynomial time of the size for the input of the problem and $1/\varepsilon$ make it possible to obtain a $(1-\varepsilon)$ or $(1+\varepsilon)$-guaranteed approximate solution (for maximization or minimization problem, respectively), where ε is an arbitrarily small positive number.

2. Full search algorithm based on the Gray codes

In 1953, physicist Frank Gray received a patent for the invention of binary reflected n-discharge codes, which were named after it [23]. Initially, these codes were used in code-pulse modulation to control various electromechanical switches and the method of analog transmission of digital signals. Currently, Gray codes are used to detect and correct errors in communication systems, control various digital sensors, encode track numbers in hard drives of computers, etc. In addition, it is known about the use of Gray codes to solve combinatorial problems "Tower of Hanoi" and "Chinese rings" [24]. More details about Gray codes can be found in the book by D. Knuth [25].

For a complete enumeration of options for solving problems (1)–(3) and (14)–(16), we will use the algorithm for generating a sequence of binary-reflected n-digit Gray codes proposed by Ehrlich [26]. The algorithm makes it possible to efficiently calculate the values of the objective function (1) or (14) and constraints (2) or (15) during the solution process. Binary-mirrored (mirrored, reflective) Gray code is defined according to the following recursive rules:

$$B_0 = \text{"},\ B_{n+1} = 0B_n 1B_n^r,\ n = 0,1,2,3,\ldots, \text{ (digits are added to the left) or}$$

$$B_0 = \text{"},\ B_{n+1} = B_n 0B_n^r 1,\ n = 0,1,2,3,\ldots, \text{ (digits are added to the right).}$$

where $B_0 = \text{"}$ – empty string, B_n – binary Gray sequence of n-bit strings, $0B_n$ and $B_n 0$ – sequence B_n prefixed with 0 at the beginning and end of each string, $1B_n^r$ and $B_n^r 1$ – sequence B_n in reverse order with a 1 prefix at the beginning and end of each line. Since the last row in B_n is equivalent to the first row in B_n^r, it is clear that at each step B_{n+1} changes exactly one bit if B_n has the same property. With each step, the length of the strings increases by 1, and their number doubles. Thus, the n-bit Gray code is an ordered cyclic sequence of 2^n n-bit strings, in which successive strings differ only by one bit. In exhaustive search algorithms for calculating the values of various functions using Gray codes, it is convenient to represent these codes as an ordered list of bit numbers that change their value to the opposite when moving from the current line to the next. This sequence of transitions P_n can be determined by the following recursive rules: $P_1 = 1,\ P_n = P_{n-1}, n, P_{n-1},\ n = 2,3,4,\ldots$. The length of the sequence P_n is equal to $2^n - 1$, and the numbering of the digits in the sequence can be performed from right to left or vice versa. For example, for $n = 4$ and the initial string 0000, $P_4 = 1,2,1,3,1,2,1,4,1,2,1,3,1,2,1$, and its length is $2^4 - 1 = 15$. The corresponding sequence of binary strings when numbering bits from left to right looks like: 0000, 1000, 1100, 0100, 0110, 1110, 1010, 0010, 0011, 1011, 1111, 0111, 0101, 1101, 1001, 0001. By numbering the digits from right to left, we get an inverted sequence corresponding to the first recursive definition of B_{n+1}. It is interesting to note that if we construct a graph whose vertices correspond to binary sequences of length n, and the edges connect two vertices that differ only by one digit, then such a graph represents a binary n-dimensional cube. Moreover, the constructed binary sequence corresponds to a Hamiltonian path in such a graph. To generate

a sequence of transitions P_n in a binary string $B = \| b_i \|$, $i = \overline{1, n+1}$, we define a vector of pointers $P = \| p_i \|$, $i = \overline{1, n+2}$, which simulates a stack for recursive definition of P_n. The optimal solution x_i^*, $i = \overline{1, n}$ will be saved in a vector $B^{opt} = \| b_i^{opt} \|$, $i = \overline{1, n+1}$, where: if $b_i^{opt} = 0$, then $x_i^* = 0$; if $b_i^{opt} = 1$, then $x_i^* = 1$; $i = \overline{1, n}$.

The dimensions of the vectors P, B and B^{opt} are increased by two and one units, respectively, due to the specifics of the algorithm. The "$\leftarrow$" sign stands for an assignment operation.

OPT1 algorithm with partial recalculation of the objective function and constraints for problem (1)–(3)

1. $CSUM \leftarrow c_1$; $ASUM \leftarrow a_1$; $CSUMOPT \leftarrow 0$; $B \leftarrow 0$; $B^{opt} \leftarrow 0$.

2. For $\{ i \mid i = \overline{1, n+2} \}$ do $p_i \leftarrow i$.

3. $i \leftarrow 1$.

4. While $i < n+1$ do steps 5-7.

5. If $b_i = 0$, then $CSUM \leftarrow CSUM - c_i$; $ASUM \leftarrow ASUM - a_i$, otherwise
$CSUM \leftarrow CSUM + c_i$; $ASUM \leftarrow ASUM + a_i$.

6. If $ASUM \le W$, then if $CSUM > CSUMOPT$ do: $CSUMOPT \leftarrow CSUM$; $B^{opt} \leftarrow B$; $ASUMOPT \leftarrow ASUM$.

7. $i \leftarrow p_1$; $b_i \leftarrow 1 - b_i$; $p_1 \leftarrow 1$; $p_i \leftarrow p_{i+1}$; $p_{i+1} \leftarrow i+1$.

8. End of the algorithm. Output values: $\max \sum_{i=1}^{n} c_i x_i^* = CSUMOPT$;
$\sum_{i=1}^{n} a_i x_i^* = ASUMOPT$; W; B^{opt}.

In the variant of solving the problem with a complete recalculation of the objective function and constraint (**OPT2 algorithm**), step 1 will be replaced by

1. $CSUMOPT \leftarrow 0$; $B \leftarrow 0$; $B^{opt} \leftarrow 0$,
and step 5 will be replaced by

5. $CSUM \leftarrow 0$; $ASUM \leftarrow 0$. For $\{ j \mid j = \overline{1, n} \}$ do $CSUM \leftarrow CSUM + c_j * b_j$; $ASUM \leftarrow ASUM + a_j * b_j$.

For an algorithm with partial recalculation of the objective function and constraints for problem (14) - (16), the pseudo code will be as follows:

1. $T_\Sigma \leftarrow 0.0$; $C_\Sigma \leftarrow 0.0$; $D_\Sigma \leftarrow 0.0$.

2. For $\{ i \mid i = \overline{1, e} \}$ do: $T_\Sigma \leftarrow T_\Sigma + t_{i1}$; $C_\Sigma \leftarrow C_\Sigma + c_{i1}$; $D_\Sigma \leftarrow D_\Sigma + \Delta c_i$.

3. $D_\Sigma \leftarrow C_\Sigma + D_\Sigma$; $T_s \leftarrow T_{\max} \times U_\Sigma$; $B \leftarrow 0$; $B^{opt} \leftarrow 0$; $C_\Sigma \leftarrow C_\Sigma + \Delta c_1$; $T_\Sigma \leftarrow T_\Sigma - \Delta t_1$.

4. For $\{ i \mid i = \overline{1, e+2} \}$ do $p_i \leftarrow i$.

5. While $i < e+1$ do steps 6-8.

6. If $b_i = 0$, then $C_\Sigma \leftarrow C_\Sigma - \Delta c_i$; $T_\Sigma \leftarrow T_\Sigma + \Delta t_i$, otherwise $C_\Sigma \leftarrow C_\Sigma + \Delta c_i$; $T_\Sigma \leftarrow T_\Sigma - \Delta t_i$.

7. If $T_\Sigma \le T_s$, then if $C_\Sigma < D_\Sigma$ do: $D_\Sigma \leftarrow C_\Sigma$; $B^{opt} \leftarrow B$; $t_{av} \leftarrow T_\Sigma / U_\Sigma$.

8. $i \leftarrow p_1$; $b_i \leftarrow 1 - b_i$; $p_1 \leftarrow 1$; $p_i \leftarrow p_{i+1}$; $p_{i+1} \leftarrow i+1$.

9. End of the algorithm. Output values: min $C_\Sigma = D_\Sigma$; t_{av}; B^{opt}.

3. Experimental comparison of OPT1 and OPT2 algorithms

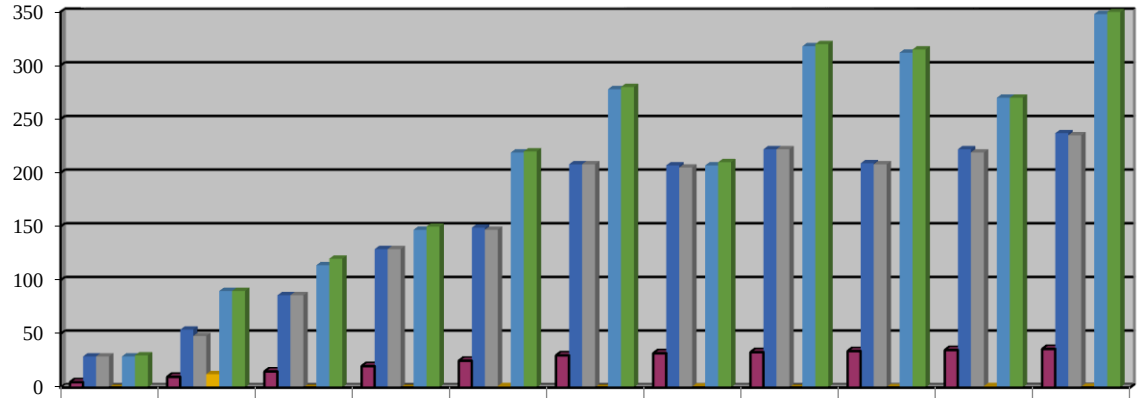
Comparison of the performance of algorithms OPT1 and OPT2 was carried out for problems (1)–(3) and (14)–(16) using examples generated by a pseudo-random number generator. Here, the results of the solution are given only for problem (1)–(3), since data similar in characteristics were obtained for problem (14)–(16). For all dimensions of the problem (1)–(3), which varied from $n=5$ to $n=36$, the cost of items c_i and their weight a_i were generated in the range from 5 to 10 and from 1 to 20, respectively. During the experiment, the accuracy of solving problem (1)–(3) by the "greedy" heuristic algorithm OPT3 with time complexity was also checked $O(n^2)$. The algorithm is based on the preliminary of ordering of items on not increasing their specific costs c_i / a_i, $i=\overline{1,n}$ in a given set with followed by the choice of items in knapsack until a restriction on size the knapsack W. Is well known, see for example [5, 27], that solutions OPT3 = $\max \sum_{i=1}^{n} c_i x_i^*$, obtained by the "greedy" algorithm, may differ from the optimal no more than twice, when choosing the final cost of the knapsack from the condition $\max \sum_{i=1}^{n} c_i x_i^* = \{\max \sum_{i=1}^{n} c_i x_i^*, \max c_i\}$, where $\max c_i$, $i=\overline{1,n}$ – is the maximum cost of the item in the specified set.

In fig. 1 shows the time to solve problem (1)–(3) (with an accuracy of two signs after the comma) on the PC with a clock frequency 2.66 GHz for OPT1 and OPT2 algorithms with partial and complete recalculation of the target function and restrictions. As can be seen from fig. 1, OPT1 algorithm can be used for practical calculations in branching schemes when the number of variables in branching nodes does not exceed 35 (calculation time at $n=35$ is about 8 minutes). OPT1 algorithm for variants 5–10 is faster than OPT2 algorithm on average in 7 times.

	1	2	3	4	5	6	7	8	9	10	11
n	5	10	15	20	25	30	32	33	34	35	36
T1	0	0	0	0.02	0.66	14.9	60.89	120.32	248.06	455.18	1751
T2	0	0	0	0.09	4.37	107.84	462.45	950.97	1964.46	3994.53	8191.27

FIG. 1. The time to solve the problem in seconds with partial (T1, OPT1 algorithm) and full (T2, OPT2 algorithm) recalculation of the target function and restrictions

In fig. 2 shows the results of the solution of problem (1)–(3) with an exact OPT1 algorithm and the "greedy" algorithm OPT3 (the results of solutions of the OPT1 and OPT2 algorithms of course coincide).



	1	2	3	4	5	6	7	8	9	10	11
n	5	10	15	20	25	30	32	33	34	35	36
OPT1	29	54	86	129	149	208	207	222	209	222	237
OPT3	29	48	86	129	147	208	205	222	208	219	235
DEL%	0	12.5	0	0	1.36	0	0.98	0	0.48	1.37	0.85
ASUMOPT	29	90	114	147	219	278	207	318	312	270	348
W	30	90	120	150	220	280	210	320	315	270	350

FIG. 2. The results of optimal (OPT1) and approximate (OPT3) solving of problem

Values $\sum_{i=1}^{n} a_i x_i^* = ASUMOPT$ are shown for OPT1 algorithm. The values of the target function obtained by the "greedy" algorithm differ from the optimal within DEL% from 0 to 12.5%. The OPT3 algorithm has a polynomial estimate of time complexity and can be applied in practice to solve problems (1)–(3) of a large dimension, when needs to quickly get the approximate value of the target function with limited computing resources. For example, the problem (1)–(3) was solved for $n=10000$ with the same boundaries of changes in values c_i and a_i in 0.11 seconds. In this case, OPT3 = 70598, ASUMOPT = 103985 at W = 104000.

All programs are written in the Fortran language (see Appendix) in the Microsoft Developer Visual Studio environment and can be adapted to work in the parallel programming system Intel® Parallel Studio XE 2020 in which are included the latest versions of C/C++ and Fortran compilers (https://software.intel.com/ru-ru/try-buy-tools). A video presentation of the work can be found at https://www.youtube.com/watch?v=4YWRO5QTQYM.

Conclusions

Application of binary-reflected (mirror, reflexive) Gray codes to solve combinatorial problems with pseudo-Boolean functions (polynomials from Boolean variables) is considered. A recursive Ehrlich algorithm is given for generating a sequence of lines n-bit Gray codes, in which each next line differs from the previous one by only one bit. On example of the 0-1 Knapsack Problem is shown how these codes can be used to efficiently compute the values of the objective function and constraints.

The purpose of the article is to show the developers of algorithms and programs how to apply Gray codes in various branching schemes of the decision algorithm, for example, in the branch and bound method, when the number of binary (Boolean) variables at the nodes of the tree is small (less than 35).

The research methodology is based on a computational experiment for solving the 0-1 knapsack problem with the proposed algorithm of exhaustive search the solution with partial and full recalculation of the

values of objective function and constraint of the problem. During the experiment, the accuracy of solving the problem by a “greedy” heuristic algorithm with time complexity $O(n^2)$ was also checked.

As a result of the experiment, it was found that the algorithm with a partial recalculation of the objective function and restrictions can be used for practical calculations in branching schemes, when the number of variables in the nodes of the branching tree does not exceed 35. The algorithm with partial recalculation is faster than the algorithm with full recalculation on average by 7 times. The heuristic "greedy" algorithm can be applied in practice to solve the 0-1 problem of a knapsack of large dimension (more than 10,000 items), when need to obtain an approximate value of the objective function at the limited computing resources.

The novelty of the work lies in the proposed approach to solving combinatorial optimization problems with pseudo-Boolean functions using Gray codes. The efficiency of the proposed algorithm with a partial recalculation of the values of the objective function and constraints is shown, and its can be applied in practice in various branching schemes of the decision algorithm.

Appendix. Text of Program "Computer program solution 0-1 Knapsack problem using reflexive Gray codes"

The program is designed for a exact and approximate solution 0-1 Knapsack Problems, 0-1 KP. For an exact solution (with complete search for solutions), the recursive algorithm of G. Erlich Generation of binary-reflected (reflexive, mirror) Gray codes is used. For an approximate solution, a "greedy" algorithm was applied, based on streamlining in descending relations of prices of items to their weights. The program can be practically used for an approximate decision 0-1 KP of the large dimension (more than 10,000 items) and for the exact solution when the number of items does not exceed 35. In addition, it can be used in branching schemes, for example, in the branch and bound algorithm, when the number the variables in the branching nodes of the decisive algorithm does not exceed 35. The program is designed to familiarize potential users with the technique of applying Gray codes when solving combinatorial problems with pseudo-Boolean functions.

The program can be used as a subprogramme to solve a wide range of combinatorial optimization problems associated with the optimization of resource allocation in various fields of economics, planning and management of business enterprises, etc.

The program consists of a main module that includes a function procedure for generating source data and an external module to solve 0-1 KP. The modules are compiled together and combined by the editor of the links into a single boot module.

In the main module of the program in dialogue mode are input: the dimension of the problem n; the borders of changing the cost of the items from MINVALC to MAXVALC; the borders of changing weights of items from MINVALA to MAXVALA; the parameter controlling the input and output data.

Next, using the pseudo-random number sensor (built-in language function $RAND()$), are generated costs and weights in the specified limits, are input the size of the Knapsack, all the arrays needed to solve the problem are generated. All operational memory for the program is allocated and deallocated in the main program module. The operation time of the program is fixed by the built-in module $cpu\_time(T)$ directly before the entrance and after exiting the external module for solving the problem.

All output can be displayed on the PC screen and into the OUT1 data set.

The program works in cyclic mode. After receiving a solution for a given dimension of the problem, you can select new parameters and continue solving the problem.

The program is launched by calling .exe module from the current directory.

```
program Test Gray codes and 0-1 Knapsack problem
use dflib
integer(4)    N,    &    ! The dimension of the problem and the dimension of the Gray Code
```

```
                B,     &    ! Restriction on the weight of the Knapsack
                I,     &    ! Variable
                 MINVALC, &   ! Lower  Border of Prices
                 MINVALA, &   ! Lower Border of Weight
                 MAXVALC, &   ! Top Border of the price of the subject
                 MAXVALA, &   ! Top Border of the weight of the subject
                 MINF,  &     ! The minimum weight of the subject
                 MAXF,  &     ! The maximum of the subject
                 SUM          ! The sum of weights items

   real(4) TS,TM,T,T1,T2,&     ! Problem solving time
                 RES           ! Variable
   integer(4),allocatable:: C(:), &  ! Vector of prices objects (N)
                            A(:)     ! Vector of weight objects (N)
   character(1)     REP

   print*,'Testing program of solving the 0-1 knapsack problem'
   212  print*,'To duplicate  output of the data in a set OUT1 ?,reply Y/N'
   read*,REP
   if (REP/='Y'.and.REP/='N') go to 212
   if (REP=='Y' ) then
   open(10,file='out1.txt')
   else
   open(10,file='nul')
   endif
   write(10,'(1x,a/)'),'Testing program of solving the 0-1 knapsack problem'
   write(6,'(1x,a)'),'Enter dimension of problem - N<=40'
   write(10,'(1x,a)'),'Enter dimension of problem - N<=40'
   read*,N
   write(6,'(1x,a,i5)'),' N = ',N; write(10,'(1x,a,i5)'),' N = ',N
  write(6,'(1x,a)'),'Enter the boundaries of the price changes of items'
   write(10,'(1x,a)'),'Enter the boundaries of the price changes of items'
   read*,MINVALC,MAXVALC
   write(6,'(1x,a,i5)'), ' MINVALC = ',MINVALC,' MAXVALC = ',MAXVALC
   write(10,'(1x,a,i5)'),' MINVALC = ',MINVALC,' MAXVALC = ',MAXVALC
   write(6,'(1x,a)'),'Enter the boundaries of the weights changes of items'
   write(10,'(1x,a)'),'Enter the boundaries of the weights changes of items
   read*,MINVALA,MAXVALA
   write(6,'(1x,a,i5,a,i5)'),' MINVALA = ',MINVALA,' MAXVALA = ',MAXVALA
   write(10,'(1x,a,i5,a,i5)'),' MINVALA = ',MINVALA,' MAXVALA = ',MAXVALA
100  continue
   allocate (C(N),A(N))
   do I=1,N
   C(I)=RND(MINVALC,MAXVALC); A(I)=RND(MINVALA,MAXVALA)
   enddo
   SUM=0
   do I=1,N
   SUM=SUM+A(I)
```

```
    enddo
write(6,'(1x,a,i8)'),'Sum of item weights = ',SUM; write(10,'(1x,a,i8)'),'Sum of item weights = ',SUM
write(6,'(1x,a)'),'Enter the size of the knapsack - B'; write(10,'(1x,a)'),'Enter the size of the knapsack - B'
    read*,B; write(6,'(1x,a,i8)'),' B = ',B; write(10,'(1x,a,i8)'),' B = ',B
    write(6,'(1x,a)'),' For start of program, press Enter'; read*
    call CBA(N,C,A,B,SUM)
    print*, 'Design successful'
    13  write(6,'(1x,a)'), 'Test of program is completed?(Y/N)'; read*,REP
    if (REP/='Y'.and.REP/='N') go to 13; if (REP=='Y' )  goto 50
    14  write(6,'(1x,a)'), 'Change N ? (Y/N)'; read*,REP
    if (REP/='Y'.and.REP/='N') go to 14
    if (REP=='Y' ) then
    write(6,'(1x,a)'), 'Enter N'; read*,N
    write(6,'(1x,a,i5)'),' N = ',N; write(10,'(1x,a,i5)'),' N = ',N
    endif
    15  write(6,'(1x,a)'), 'Change MINVALC, MAXVALC ? (Y/N)'; read*,REP
    if (REP/='Y'.and.REP/='N') go to 15
    if (REP=='Y' ) then
    write(6,'(1x,a)'), 'Enter MINVALC, MAXVALC'; read*,MINVALC,MAXVALC
    write(6,'(1x,a,i5)'), ' MINVALC = ',MINVALC,' MAXVALC = ',MAXVALC
    write(10,'(1x,a,i5)'),' MINVALC = ',MINVALC,' MAXVALC = ',MAXVALC
    endif
    16  write(6,'(1x,a)'), 'Change MINVALA, MAXVALA ? (Y/N)'; read*,REP
    if (REP/='Y'.and.REP/='N') go to 16
    if (REP=='Y' ) then
    write(6,'(1x,a)'), 'Enter MINVALA, MAXVALA'; read*,MINVALA,MAXVALA
    write(6,'(1x,a,i5,a,i5)'),' MINVALA = ',MINVALA,' MAXVALA = ',MAXVALA
    write(10,'(1x,a,i5,a,i5)'),' MINVALA = ',MINVALA,' MAXVALA = ',MAXVALA
    endif
    deallocate (C,A)
    write(6,'(1x,a)'),'-----------------------------------------------------';
    write(10,'(1x,a)'),'-----------------------------------------------------'
    go to 100
    50  continue
    write(6,'(1x,a)'),'-----------------------------------------------------'
    write(10,'(1x,a)'),'-----------------------------------------------------'
    write(6,'(1x,a)'), 'Test of program is successfully completed !'
    write(10,'(1x,a)'), 'Test of program is successfully completed !'
    write(6,'(1x,a)'),'-----------------------------------------------------'
    write(10,'(1x,a)'),'-----------------------------------------------------'

    contains

    integer function RND(MINVAL,MAXVAL)
    RES=RAND()
    if (ceiling(RES*MAXVAL)< MINVAL) then
    RND=MINVAL+ceiling(RES*(MAXVAL-MINVAL))
    else
```

```
RND=ceiling(RES*MAXVAL)
endif
end function RND

end program Test Gray codes and 0-1 Knapsack problem

subroutine CBA(N,C,A,B,SUM)
integer(4) N,          &
           B,          &
           SUM,        &
           KI,I,RM,J,K, &
           CSUM,ASUM,MSUM,ASUMOPT,CSUMA
integer(4)   C(N)
integer(4)   A(N)
integer(4), allocatable:: SE(:),  &
                          WU(:)
byte, allocatable:: BI(:), BOPT(:)
real(4), allocatable:: WKL(:),  &
                       VUS(:)
real(4) T1,T2,T3,T33,MAX,CM
character(72)    DIS
character(1)     REP

allocate (WKL(N),VUS(N),SE(N),BOPT(N+1))
if (N<=40) then
allocate (WU(N+2),BI(N+1))
endif
do I=1,N
SE(I)=I; WKL(I)=(.1E+01*C(I))/A(I); VUS(I)=WKL(I)
enddo
call cpu_time(T1)
do I=1,N-1
MAX=VUS(I)
do J=I+1,N
if (VUS(J)>MAX) then
CM=VUS(I); RM=SE(I)
VUS(I)=VUS(J); SE(I)=SE(J)
VUS(J)=CM; SE(J)=RM
endif
enddo
enddo
CSUM=0; ASUM=0; BOPT=0
do I=1,N
if (ASUM+A(SE(I))<=B) then
CSUM=CSUM+C(SE(I)); ASUM=ASUM+A(SE(I)); BOPT(SE(I))=1
else
go to 18
endif
```

```
    enddo

    18 continue
    CSUMA=CSUM
    call cpu_time(T2)
    write(6,'(5x,a,f10.2)'),'TIME of APPROXIMATE SOLUTION = ',T2-T1
    write(10,'(5x,a,f10.2)'),'TS = ',T2-T1
    write(6,'(1x,3(a,i8))'), ' CSUM = ',CSUM,' ASUM = ',ASUM,' B = ',B
    write(10,'(1x,3(a,i8))'), ' CSUM = ',CSUM,' ASUM = ',ASUM,' B = ',B
    write(6,'(1x,a/)'), '  '
    write(6,'(1x,a,i8)'),' APPROXIMATE SOLUTION = ',CSUM
    write(10,'(1x,a,i8)'),' APPROXIMATE SOLUTION = ',CSUM

 !!! Algorithm with complete recalculation of the target function and restrictions

    if (N>40) go to 5055
    write(6,'(1x,a/)'), '  '; write(6,'(1x,a/)'), ' OPTIMAL SOLUTION 1 '
    MSUM=0; BI=0; BOPT=0
    do I=1,N+2
    WU(I)=I
    enddo
    I=1
    call cpu_time(T1)
    do while (I<N+1)
    CSUM=0; ASUM=0
    do J=1,N
    CSUM=CSUM+C(J)*BI(J); ASUM=ASUM+A(J)*BI(J)
    enddo
    if (ASUM<=B) then
    if (CSUM>MSUM) then
    MSUM=CSUM; BOPT=BI; ASUMOPT=ASUM
    endif
    endif
     I=WU(1); BI(I)=1-BI(I); WU(1)=1; WU(I)=WU(I+1); WU(I+1)=I+1
     enddo
     call cpu_time(T2)
     T3=T2-T1
     write(6,'(1x,a/)'), '  '
 write(6,'(1x,a/)'), ' OPTIMAL SOLUTION 1 '; write(10,'(1x,a/)'), ' OPTIMAL SOLUTION 1 '
     write(6,'(1x,a,i8)'), ' max CSUM = ',MSUM; write(10,'(1x,a,i8)'), ' max CSUM = ',MSUM
     write(6,'(1x,2(a,i8))'), ' ASUM = ',ASUMOPT,' B = ',B;
     write(10,'(1x,2(a,i8))'), ' ASUM = ',ASUMOPT,' B = ',B
     T1=((.1E+01*MSUM/CSUMA)-.1E+01)*100
     write(6,'(1x,a/)'), '  '
write(6,'(1x,a,f12.4,a)'),' Deviation of the approximate solution from the optimal solution = ',T1,' % '
write(10,'(1x,a,f12.4,a)'),' Deviation of the approximate solution from the optimal solution = ',T1,' % '
write(6,'(1x,a,f8.2,a)'),' T   = ',T3,  '  Time of the optimal decision 1 of a problem'
write(10,'(1x,a,f8.2,a)'),' T   = ',T3,  '  Time of the optimal decision 1 of a problem'
```

```
!!! Algorithm with partial recalculation of the target function and restrictions

if (N >40) go to 5055
write(6,'(1x,a/)'), '  '; write(6,'(1x,a/)'), ' OPTIMAL SOLUTION 2 '
CSUM=C(1); ASUM=A(1); MSUM=0; BI=0; BOPT=0
do I=1,N+2
WU(I)=I
enddo
I=1
call cpu_time(T1)
do while (I<=N+1)
if (BI(I)==0) then
CSUM=CSUM-C(I); ASUM=ASUM-A(I)
else
CSUM=CSUM+C(I); ASUM=ASUM+A(I)
endif
if (ASUM<=B) then
if (CSUM>MSUM) then
MSUM=CSUM
BOPT=BI
ASUMOPT=ASUM
endif
endif
I=WU(1); BI(I)=1-BI(I); WU(1)=1; WU(I)=WU(I+1); WU(I+1)=I+1
enddo
call cpu_time(T2)
T3=T2-T1
write(6,'(1x,a/)'), '  '
write(6,'(1x,a/)'), ' OPTIMAL SOLUTION '; write(10,'(1x,a/)'), ' OPTIMAL SOLUTION '
write(6,'(1x,a,i8)'), ' max CSUM = ',MSUM; write(10,'(1x,a,i8)'), ' max CSUM = ',MSUM
write(6,'(1x,2(a,i8))'), ' ASUM = ',ASUMOPT,' B = ',B
write(10,'(1x,2(a,i8))'), ' ASUM = ',ASUMOPT,' B = ',B
T1=((.1E+01*MSUM/CSUMA)-.1E+01)*100
write(6,'(1x,a/)'), '  '
write(6,'(1x,a,f12.4,a)'),' Deviation of the approximate solution from the optimal solution = ',T1,' % '
write(10,'(1x,a,f12.4,a)'),' Deviation of the approximate solution from the optimal solution = ',T1,' % '
write(6,'(1x,a,f8.2,a)'),' T= ',T3,'Time of the optimal decision of a problem'
write(10,'(1x,a,f8.2,a)'),' T= ',T3,'Time of the optimal decision of a problem'
5055    continue
end subroutine CBA !
```

References

1. Martello S., Toth P. Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc., 1990.
2. Drezner Z. Facility Location. A Survey of Applications and Methods. Springer, 1995.
3. Nemhauser G.L., Wolsey L.A. Integer and combinatorial optimization. John Wiley & Sons, Inc., 1999.
4. Schrijver. Combinatorial optimization. Polyhedra and efficiency. Berlin: Springer, 2003.
5. Kellerer H., Pferschy U., Pisinger D. Knapsack problems. Springer-Verlag Berlin Heidelberg, 2004. https://doi.org/10.1007/978-3-540-24777-7

6. Korte B., Vygen J. Combinatorial Optimization. Theory and Algorithms. Springer, 2005.
7. Hifi M., Michrafy M., Sbihi A. Heuristic algorithms for the multiple-choice multidimensional knapsack problem. *J. of Oper. Res. Soc*. 2004. **55** (12). P. 1323–1332. https://doi.org/10.1057/palgrave.jors.2601796
8. Akbar M.M., Rahman M.S., Kakobad M., Manning E.G., Shoja G.C. Solving the Multidimensional Multiple-choice Knapsack Problem by constructing convex hulls. *Comp. and Oper. Res*. 2006. **33** (5). P. 1259–1273. https://doi.org/10.1016/j.cor.2004.09.016
9. Shahriar Z., Akbar M.M., Rahman M.S., Newton M. M. A multiprocessor based heuristic for multi-dimensional multiple-choice knapsack problem. *The J. of Supercomputing*. 2008. **43** (3). P. 257–280. https://doi.org/10.1007/s11227-007-0144-2
10. Lazarev A.A., Werner F. A graphical realization of the dynamic programming method for solving NP-hard combinatorial problems. *Computers & Mathematics with Applications*. 2009. **58** (4). P. 619–631. https://doi.org/10.1016/j.camwa.2009.06.008
11. Leblet H.J., Simon G. Hard multidimensional multiple choice knapsack problems: an empirical study. *Comp. and Oper. Res*. 2010. **37** (1). P. 172–181. https://doi.org/10.1016/j.cor.2009.04.006
12. Posypkin M.A., Sin S.T.T. Comparative analysis of the efficiency of various dynamic programming algorithms for the knapsack problem. *Computer Science. 2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW).* 2016. https://doi.org/10.1109/EIConRusNW.2016.7448182
13. Gary M.R., Johnson D.S. Computers and intractability: A guide to the theory of NP-completeness. W. H. Freeman & Co. New York, NY, USA, 1979.
14. Hammer P.L., Rudeanu S. Boolean Methods in Operations Research and Related Areas. Berlin, Springer-Verlag; New York, Heidelberg, 1968. https://doi.org/10.1007/978-3-642-85823-9
15. Beresnev V.L., Ageev A.A. Minimization Algorithms for Some Classes of Polynomials in Boolean Variables. *Modeli i metody optimizatsii*. 1988. No. 10. P. 5–17. (in Russian)
16. Boros E., Hammer P.L. Pseudo-Boolean Optimization. *Discrete Applied Mathematics*. 2002. **123** (1–3). P. 155–225. https://doi.org/10.1016/S0166-218X(01)00341-9
17. Crama Y., Hammer P.L. Boolean Functions: Theory, Algorithms, and Applications. New York, Cambridge University Press, 2011. https://doi.org/10.1017/CBO9780511852008
18. Vasyanin V.A., Ushakova L.P. Gray codes in combinatorial optimization problems. *Matematicheskoye modelirovaniye v ekonomike*. 2019. No. 1–2. P. 63–69. (in Russian).
19. Trofymchuk O.M., Vasyanin V.A. Choosing the Capacity of Arcs with Constraint on Flow Delay Time. *Cybernetics and Systems Analysis*. 2019. **55** (4). P. 561–569. https://doi.org/10.1007/s10559-019-00165-0
20. Trofymchuk O.M., Vasyanin V.A. Simulation of Packing, Distribution and Routing of Small-Size Discrete Flows in a Multicommodity Network. *Journal of Automation and Information Sciences*. 2015. **47** (7). P. 15–30. https://doi.org/10.1615/JAutomatInfScien.v47.i7.30
21. Vasyanin V.A. Problem of Distribution and Routing of Transport Blocks with Mixed Attachments and Its Decomposition. *Journal of Automation and Information Sciences*. 2015. **47** (2). P. 56–69. https://doi.org/10.1615/JAutomatInfScien.v47.i2.60
22. Vasyanin V.A., Trofymchuk O.M., Ushakova L.P. Economic-mathematical models of the problem of distribution of flows in a multicommodity communication network. *Matematicheskoye modelirovaniye v ekonomike*. 2016. No. 2. P. 5–21. (in Russian)
23. Gray F. Pulse code communication. U.S. Patent 2632058, March 17, 1953.
24. Gardner M. Mathematical Puzzles and Entertainment: 2nd ed., corrected and supplemented / Translation from English. Moscow: "Peace", 1999. (in Russian)
25. Knuth E. The Art of Computer Programming. Volume 4A / Combinatorial Algorithms. Part 1. Addison Wesley Longman, Inc., 2011.
26. Bitner J.R., Ehrlich G., Reingold E.M. Efficient Generation of the Binary Reflected Gray Code and its Applications. *Comm. ACM*. 1976. No. 19. P. 517–521. https://doi.org/10.1145/360336.360343
27. Kuzurin N.N., Fomin S.A. Effective algorithms and computing complexity. M.: MPTI, 2008. (in Russian)

Received 19.11.2022

Trofymchuk Oleksandr,
Doctor of Technical Sciences, Professor, Corresponding Member of the NAS of Ukraine, Director,
Institute of Telecommunications and Global Information Space of the NAS of Ukraine, Kyiv,
https://orcid.org/0000-0003-3358-6274
itgis@nas.gov.ua

Vasyanin Volodymyr,
Doctor of Technical Sciences, Chief of department,
Institute of Telecommunications and Global Information Space of the NAS of Ukraine, Kyiv,
https://orcid.org/0000-0003-4046-5243
archukr@meta.ua

Sokolov Volodymyr,
PhD in IT, Associate Professor,
Borys Grinchenko Kyiv University, Ukraine,
https://orcid.org/0000-0002-9349-7946
v.sokolov@kubg.edu.ua

Arkadii Chikrii,
Doctor of Science (phys. & math.), Professor, Academician of the NAS of Ukraine, Chief of department,
V.M.Glushkov Institute of Cybernetics of the NAS of Ukraine, Kyiv,
https://orcid.org/0000-0001-9665-9085
g.chikrii@gmail.com

Liudmyla Ushakova,
Leading engineer,
Institute of Telecommunications and Global Information Space of the NAS of Ukraine, Kyiv.
https://orcid.org/0000-0002-9020-1329
archukr@i.ua

UDC 519.168

Oleksandr Trofymchuk [1], Volodymyr Vasyanin [1 *], Volodymyr Sokolov [2], Arkadii Chikrii [3], Liudmyla Ushakova [1]

On the Use of Gray Codes for Solving 0-1 Combinatorial Problems of Optimization in Environmental and Economic Systems

[1] Institute of Telecommunications and Global Information Space of the NAS of Ukraine, Kyiv
[2] Borys Grinchenko Kyiv University, Ukraine
[3] V.M.Glushkov Institute of Cybernetics of the NAS of Ukraine, Kyiv
[*] Correspondence: archukr@meta.ua

Introduction. The application of binary-reflected (mirror, reflexive) Gray codes for solving combinatorial problems with pseudo-Boolean functions (polynomials from Boolean variables) is considered. A recursive Ehrlich algorithm is given for generating a sequence of lines n-bit Gray codes, in which each subsequent line differs from the previous one by only one digit (bit). As an example of the effectiveness of the use of these codes, the solution of two combinatorial problems with Boolean variables with a complete enumeration of solutions is considered, and it is shown how these codes can be used to efficiently calculate the values of the objective function and constraints. The results of an experimental study are presented, which show that Gray codes can be practically applied in branching schemes, for example, in the branch and bound method, when the number of variables in the branching nodes of the decision algorithm does not exceed 35.

Purpose. The purpose of the article is to show the developers of algorithms and programs how to apply Gray codes in various branching schemes of the decision algorithm, for example, in the branch and bound method, when the number of binary (Boolean) variables at the nodes of the tree is small (less than 35).

The technique. The research methodology is based on a computational experiment for solving the 0-1 knapsack problem with the proposed algorithm of exhaustive search the solution with partial and full recalculation of the values of objective function and constraint of the problem. During the experiment, the accuracy of solving the problem by a "greedy" heuristic algorithm with time complexity $O(n^2)$ was also checked.

Results. As a result of the experiment, it was found that the algorithm with a partial recalculation of the objective function and restrictions can be used for practical calculations in branching schemes, when the number of variables in the nodes of the branching tree does not exceed 35. The algorithm with partial recalculation is faster than the algorithm with full recalculation on average by 7 times. The heuristic "greedy" algorithm can be

applied in practice to solve the 0-1 problem of a knapsack of large dimension (more than 10,000 items), when need to obtain an approximate value of the objective function at the limited computing resources.

Scientific novelty and practical significance. The novelty of the work lies in the proposed approach to solving combinatorial optimization problems with pseudo-Boolean functions using Gray codes. The efficiency of the proposed algorithm with a partial recalculation of the values of the objective function and constraints is shown, and its can be applied in practice in various branching schemes of the decision algorithm.

Keywords: Gray codes, combinatorial optimization problems, problem solving time.

УДК 519.168

О. Трофимчук [1], В. Васянін [1*], В. Соколов [2], А. Чикрій [3], Л. Ушакова [1]

Про використання кодів Грея для розв'язування 0-1 комбінаторних задач оптимізації в еколого-економічних системах

[1] Інститут телекомунікацій і глобального інформаційного простору НАН України, Київ
[2] Київський університет імені Бориса Грінченка, Україна
[3] Інститут кібернетики імені В.М. Глушкова НАН України, Київ
[*] Листування: archukr@meta.ua

Вступ. Розглядається застосування двійково-відображених (дзеркальних, рефлексивних) кодів Грея для розв'язання комбінаторних задач з псевдобулевими функціями (поліномами від булевих змінних). Наводиться рекурсивний алгоритм Ерліха для генерації послідовності рядків *n*-розрядних кодів Грея, в якій кожний наступний рядок відрізняється від попереднього тільки одним розрядом (бітом). Як приклад ефективності використання цих кодів розглянуто розв'язання двох комбінаторних задач з булевими змінними з повним перебором розв'язків, показано, як ці коди можна використовувати для ефективного обчислення значень цільової функції і обмежень. Наведено результати експериментального дослідження, які показують, що коди Грея можуть бути практично застосовані у схемах розгалуження, наприклад, у методі гілок і меж, коли кількість змінних у вузлах розгалуження вирішального алгоритму не перевищує 35.

Мета. Робота полягає у тому, щоб показати розробникам алгоритмів і програм як можна застосувати коди Грея у різних схемах розгалуження вирішального алгоритму, наприклад, у методі гілок і меж, коли кількість двійкових (булевих) змінних у вузлах дерева розгалуження невелика (менше 35).

Методика. Дослідження грунтуються на проведенні обчислювального експерименту розв'язання 0-1 задачі про ранець запропонованим алгоритмом перебору варіантів розв'язку з частковим і повним перерахунком значень цільової функції і обмеження задачі. При проведенні експерименту перевірялася також точність розв'язання задачі «жадібним» евристичним алгоритмом з часової складністю $O(n^2)$.

Результати. В наслідок проведеного експерименту встановлено, що алгоритм з частковим перерахунком цільової функції і обмеження може застосовуватися для практичних розрахунків у схемах розгалуження, коли кількість змінних у вузлах дерева розгалуження не перевищує 35. Алгоритм з частковим перерахунком швидше алгоритму з повним перерахунком у середньому в 7 разів. Евристичний «жадібний» алгоритм можна застосовувати на практиці для розв'язання 0-1 задачі про ранець великої розмірності (більше 10000 предметів), коли достатньо отримати наближене значення цільової функції при обмежених обчислювальних ресурсах.

Наукова новизна і практична значимість. Новизна роботи полягає у запропонованому підході до розв'язання комбінаторних задач оптимізації з псевдобулевими функціями з використанням кодів Грея. Показана ефективність запропонованого алгоритму з частковим перерахунком значень цільової функції і обмежень може застосовуватися на практиці в різних схемах розгалуження вирішального алгоритму.

Ключові слова: коди Грея, задачі комбінаторної оптимізації, час розв'язання задачі.