STATE UNIVERSITY OF MOLDOVA

Dmitri TERZI

NUMERICAL METHODS OF DISCRETE OPTIMIZATION

Chisinau, 2024
CEP USM

CZU 519.6
T 45

Monografie a fost aprobată spre editare la ședința Consiliului Științific USM

Recenzenți: Vasile Moraru, dr., prof. univ.; Ion Jardan, dr., conf. univ.
Andrei Poștaru, dr., conf. univ.

Monografia prezintă metode numerice de optimizare discretă pentru rezolvarea problemelor de tip transport, inclusiv programarea liniară cu numere întregi, problema alegerii și problema comis-voiajorului. Va fi util specialiştilor implicaţi în aplicarea metodelor matematice în economie, studenţilor care studiază matematica aplicată şi informatică. Monografia a fost întocmită ca urmare a implementării proiectului postdoctoral „Elaborarea metodelor de rezolvare a problemelor de optimizare discretă de tip transport” cu cifrul 23.00208.5007.09/PD II, finanțat de către ANCD.

DESCRIEREA CIP A CAMEREI NAŢIONALE A CĂRŢII DIN REPUBLICA MOLDOVA

Terzi, Dmitri.
Numerical methods of discrete optimization /Dmitri Terzi; State University of Moldova. – Chişinău : Editura USM, 2024. – 180 p. : fig., tab.
Bibliogr.: p. 170-179 (155 tit.). – 50 ex.
ISBN 978-9975-62-830-3.
519.6
T 45

ISBN
978-9975-62-830-3.

© Dmitri TERZI
© Editura USM, 2024

CONTENTS

PREFACE 5
INTRODUCTION. 7
DISCRETE OPTIMIZATION PROBLEMS AND METHODS FOR SOLVING THEM 7

CHAPTER 1. ON ONE CLASS OF PROBLEMS WITH SEPARATING SETS OF CONSTRAINTS 24
1.1 On the definition and solution of problems with separating sets of constraints 24
1.2 Modification of the cutting method 26
1.3 Hermite's algorithm for solving systems of linear equations in integers 35

CHAPTER 2. APPROXMATE SOLUTIONS OF PROBLEMS WITH SEPARATING SETS OF CONSTRAINTS 41
2.1 Iterative process of solving problems with boolean variables. 44
2.1.1 Description of the iterative process for solving the problem T_s 45
2.1.2 Convergence Theorem. 50
2.1.3 On assessing the effectiveness of the iterative process 55
2.2 Vector generators *GV1-GV8*. 56
2.3 Transformations of the integer linear programming problem 61
2.3.1 Reduction of a general integer linear programming problem to a series of problems with positive coefficients. 61
2.3.2 On aggregation of equations 65
2.4 An approximate algorithm for solving an integer programming problem with positive coefficients. 68

CHAPTER 3. ALGORITHMS AND PROGRAMS FOR DISCRETE AND COMBINATORIAL OPTIMIZASTYION 74
3.1 Search Algorithms for a System of Different Representatives of Subsets of a Finite Set 74
3.2 Criteria for the Existence of *SDR* a Family of Subsets 76
3.3 Algorithms for Constructing *SDR* 77
3.4 Algorithm 3. Search for *SDR* using Hall's theorem 79
3.5 Algorithm 4. Reducing *SDR* search to solving an assignment problem 82
3.6 Algorithm 5. *SDR* search associated with the construction of cycles 87
3.7 Method of searching for *SDR* by incidence matrix 88
3.8 On the extension of the method for solving the choice problem 90
3.9 Heuristic algorithms 91
3.9.1 Decription of heuristics 92
3.9.2 The experimental part 96

3.10 Statistical method of approximate solution of the discrete optimization problem 99
3.11 Decomposition of information relative to bypass points by convex hulls 106
3.11.1 On the solution of special traveling salesman problems. 107
3.11.2 On the solution of the TSP using the data decomposition algorithm for convex hulls. 107

CHAPTER 4. A NATURAL METHOD FOR SOLVING THE TRAVELING SALESMAN PROBLEM 116
4.1 An approach to solve the traveling salesman problem with help of the potentials method 116
4.2 Methods for constructing cyclic solutions 120
4.2.1 Construction of initial feasible cyclic solutions based on the cost matrix and the evaluation matrix 127
4.2.2 On the procedure for constructing orthogonal cyclic permutations and its application 129
4.3 Application of potentials method technology for solving the traveling salesman problem. 134
4.4 On one optimality criterion for solving the traveling salesman problem 145
4.5 On one implementation of a natural approach to solving the traveling salesman problem 147
4.6 Modification of the potentials method in solving the traveling salesman problem 148
4.6.1 Description of algorithms and functions 149
4.6.2 Rationale for algorithms 151
4.7 A new approach to solving the traveling salesman problem 155
4.7.1 Optimality criteria 156
4.7.2 Solution of the traveling salesman problem as a special problem of the transport type 157
4.8 Construction of the traveling salesman problem with a predetermined optimal solution. 159
4.8.1 Algorithm for constructing the traveling salesman problem with a given optimal solution 160
4.8.2 General case of constructing a traveling salesman problem with a given optimal solution using the function c=NovGenerTSP (n,xo,yo,mc,k) 167
REFERENCES 170

PREFACE

The main purpose of writing this monograph is to provide readers with certain new opportunities for understanding and applying discrete optimization methods in solving numerous practical problems that arise in various areas of human activity.

The book is developed mainly on the basis of the author's research and consists of four chapters. The first two chapters contain numerical algorithms for linear integer programming, which were used to solve specific problems reflected in the relevant works from the list of references. The third and fourth chapters relate to another area of discrete optimization. Here, methods for finding a system of different representatives are presented, and methods for solving the choice problem are applied. Particular attention is paid to methods for solving the traveling salesman problem. The choice of such a topic for research and writing the work is explained by the desire to show the difficulties and ways to overcome them in solving discrete problems.

The work is intended for specialists in the application of numerical methods for solving discrete optimization problems, in particular, transport-type problems, for students of applied mathematics studying optimization methods. Some algorithms are brought to programs, which allows the user to better understand the new original methods available in the monograph, for example, for searching for systems of different representatives of subsets of a finite set, solving the traveling salesman problem using operations three-element of replacement, methods for generating cyclic solutions based on cost and estimate matrices. To understand the structure of the problem, methods for constructing a traveling salesman problem with a given cyclic solution as optimal are presented, which is an impetus for new research and conducting mass computational experiments

INTRODUCTION. DISCRETE OPTIMIZATION PROBLEMS AND METHODS FOR SOLVING THEM

Many problems in science and economics require an integer solution. In some problems, the sought variables may be indivisible objects, such as the number of units of production, sets of equipment, the number of machines in a control complex, vehicles, etc. In other problems, integerness arises when choosing design options. Integer problems also include problems of rational cutting of materials, the parameters of which are whole sheets of plywood, tin, glass, etc. [1-10,11-15,17-21,23-27],[48,80,118,130,134,138].

Problems of finding the conditional extremum of a linear function with the requirement of integerness of all (or part) variables are called fully (or partially) integer programming. These problems are divided into two classes: fully integer and mixed (partially) integer [80,87,98,110, 113,117,129,136].

A trivial example of an integer programming problem is the classical linear programming transportation problem [149]. If the initial quantities of manufactured and consumed products (deliveries) are expressed as integers, then the condition of integer values of the variables in the optimal plan is satisfied. When the initial data are integer, various variants of the transport problem also have an integer solution, for example, the assignment problem, the problem of flows in networks.

Integer problems can also be classified by the value of the variables. For example, integer problems can take the values 0 or 1 [28,111,153]. Thus, many combinatorial problems with the help of artificial so-called Boolean variables, which take the value 1 if some events occur and 0 if these events do not occur, can be written in terms of integer linear programming . These include the traveling salesman problem, which describes a class of models for finding routes for cargo delivery that minimize the total mileage; scheduling theory problem; covering problem (network optimization).

Many problems of optimal development, specialization and placement of production are practically solved by using integer programming [3,8,13,71,74].

The integer linear programming problem is formulated as follows: find a solution (plan) $x = (x_1, x_2, ..., x_n)$ for a problem with an objective function

$$f = \sum_{j=1}^{n} c_j x_j \to \min(\max) \tag{1}$$

that satisfies the restrictions

$$\sum_{j=1}^{n} a_{ij} x_j w_i b_i,\ i = \overline{1,m}; \tag{2}$$

$$x_j \geq 0,\ j = \overline{1,n}; \tag{3}$$

$$x_j \text{-} intrger,\ j = \overline{1,n_1},\ n_1 \leq n. \tag{4}$$

Problem (1)-(4) without constraints (4) is a general linear programming problem in a continuous setting. If $n_1 = n,$ then problem (1)-(4) is called a completely integer problem; if $n_1 < n$, then a partially integer problem.

Problem (1)-(2) with condition

$$x_j = 0 \vee 1,\ j = \overline{1,n}; \tag{5}$$

is a special case of an integer problem and is called a problem with boolean variables.

Let us consider the problem of planning the production of indivisible types of products.

Let $i = 1, 2, ..., m$ denote production factors; let $j = 1, 2, ..., n$ denote types of final products. Let us further denote:

a_{ij} - the amount of resource of type i required to produce a unit of product of type j;

b_i - available resources of type i;

c_j - profit obtained from the sale of a unit of product of type j.

Let products of type j be indivisible. Let us assume that it is necessary to draw up a production program that ensures a minimum total profit, given the limited availability of these resources. By designating the desired objects of manufactured products as x_j, we see that this problem is reduced to model (1)-(4).

Let us consider the problem of planning the use of indivisible production factors. Let $i = 1, 2, ..., m$ denote the types of work to be performed; let $j = 1, 2, ..., n$ denote the types of equipment.

Let a_{ij} - be the cost of using a unit of equipment of type j on work of type i;

b_i - restrictions on the total cost for work of type i;

c_j - the effect of using a unit of equipment of type j.

Let all types of equipment be used only in whole non-negative quantities. It is required to draw up a program for using the equipment that satisfies the cost constraints and gives the maximum total effect. Let x_j denote the required number of units of equipment of type j. Then the problem will be reduced to model (1)-(4).

This model may also contain upper limits on the number of units of equipment of each type, for example, non-negative integers k_j, $j = 1, 2, ..., n$; then the conditions of problem (1)-(4) should be supplemented with additional restrictions

$$x_j \le k_j, \ j = \overline{1, n}.$$

One of the earliest integer problems that stimulated the development of integer programming is the knapsack problem (also called the backpack problem).

There are *n* objects; the following values are given:

a_j - weight of object *j*;

c_j - value of object *j*.

It is required to load a backpack of volume v with a set of items with the maximum total value.

If we introduce variables x_j, $j = 1, 2, ..., n$, that have the following meaning:

$$x_j = \begin{cases} 0, \ if \ \ item \ j \ \ is \ subject \ to \ loading; \\ 1 - otherwise, \end{cases}$$

then the knapsack problem will be reduced to maximizing the objective function *f*, i.e. find

$$f = \sum_{j=1}^{n} c_j x_j \to \max$$

under conditions (5) and

$$\sum_{j=1}^{n} a_j x_j \le v. \qquad (6)$$

Other versions of this model may include restrictions (6) of several types, for example, not only the total weight of the loaded items may be limited, but also their volume, etc.

Of practical interest are combinatorial problems, which we will understand as any formulation where, among a finite number of options, it is necessary to make a choice that gives the minimum (or maximum) value of the objective function [23,63-155].

Consider a simple combinatorial problem known as the assignment problem (or choice problem).

Let there be n types of work and n candidates for these works. The assignment of candidate i to work j is associated with costs c_{ij} $(i, j = 1,2,...,n)$. It is required to find the assignment of candidates to all works that yield minimal total costs. In this case, each candidate can be assigned to only one work, and each work can be occupied by only one candidate.

The solution to this problem is a permutation of n numbers, which for large numbers n is practically impossible to search directly. The problem can be reduced to linear programming. The objective function of the problem is defined on the set of permutations of numbers $(1,2,...,n)$; each such permutation is described by a point in n -dimensional space; this point is conveniently represented as a matrix $[x_{ij}]$ of size $n \times n$. The elements x_{ij},

$$x_{ij} = \begin{cases} 1, \ if \ the \ i-th \ applicant \ is \ appointed \ to \ the \ j-th \ job; \\ 0-otherwise; \end{cases} \tag{7}$$

matrices $[x_{ij}]$ must be subject to the conditions:

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

$$\sum_{i=1}^{n} x_{ij} = 1, \ \ j = \overline{1,n}. \tag{9}$$

Condition (8) means that each applicant can be assigned to only one job, condition (9) means that each job is intended for only one applicant.

It is required to find numbers x_{ij} that satisfy conditions (7)-(9) and minimize total costs, i.e.

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \to \min. \tag{10}$$

As can be seen, the assignment problem (7)-(9) is a special case of the transportation problem (in which $m = n,\ a_i = b_j = 1$).

Let us give the assignment problem another substantive meaning.

Let there be n aircraft of different types that need to be distributed among n airlines. The expected effect c_{ij} from using the i -th aircraft on the j -th airline is known, measured, for example, by the number of passengers served.

The task is to assign aircraft to the lines (one for each line) in such a way that the total effect of using all aircraft is maximal (i.e., the number of passengers served is maximized, for example).

If we introduce variables x_{ij} defined by the formula

$$x_{ij} = \begin{cases} 1, \textit{ if the } i\textit{-th plane is assigned to the } j\textit{-th line}; \\ 0 - \textit{otherwise}; \end{cases}$$

then the mathematical model can be written as follows.

Find such values of variables x_{ij} that give the value of the objective function

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \to \min$$

(cumulative effect) under conditions

$$\sum_{i=1}^{n} x_{ij} = 1,\ \ j = \overline{1,n}$$

(an aircraft is assigned to each line);

$$\sum_{j=1}^{n} x_{ij} = 1,\ \ i = \overline{1,n}$$

(each plane is assigned to one line).

Condition $x_{ij} = 1$ or $x_{ij} = 0$, generally speaking, greatly complicates the task.

As for the assignment problem under consideration, it has been proven that condition $x_{ij} = 1 \vee 0$ can be replaced by a simpler one, namely $0 \le x_{ij} \le 1$. More precisely, if we apply the algorithm for solving the transportation problem as usual, then in this case it will automatically lead to integers x_{ij}, i.e. to solving the problem in its original formulation. Thus, it turns out that the assignment problem is also essentially described by the model of the transportation problem. (However, for the assignment problem there is also a simpler algorithm for solving the general transportation problem).

The traveling salesman problem, which has great practical significance, has become widely used.

There are n points. A matrix $[c_{ij}]$ of transport distances between these points is given, with each point connected to all the others. Leaving one point, a salesman must visit the other points once and return to the starting point, while he forms a closed route (without loops). In what order should the points be visited so that the distance traveled is minimal?

As in the assignment problem, we are talking about minimizing the total distance (costs). The salesman must visit each point only once (compare with the assignment problem; in it, each candidate is appointed to only one position); the problem differs in that it is necessary to construct a closed route.

Let's introduce variables

$$x_{ij} = \begin{cases} 1, \textit{ if a salesman from point } i \textit{ moves to point } j; \\ 0 - \textit{otherwise}, \end{cases}$$

where $i, j = 1, 2, ..., n; \ i \neq j$.

It is necessary to minimize

$$f = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

under restrictions

$$\sum_{i=1}^{n} x_{ij} = 1, \ \ j = \overline{1, n}; \tag{11}$$

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

$$t_i - t_j + n x_{ij} \leq n - 1, \ \ i, j = 1, 2, ..., n; \ \ i \neq j. \tag{13}$$

Here t_i and t_j are arbitrary real values. Constraint (11) means that the salesman leaves each point only once; constraint (12) means that the salesman enters any point only once; constraint (13) ensures that the route containing n points is closed and has no loops.

If the traveling salesman problem describes a class of models for finding a route for cargo delivery that minimizes the total mileage, then the scheduling theory problem is a model for the bulk of production organization problems. The latter is also called the calendar planning problem.

In general, it is described as follows. There are machines m and n parts. Each part must be processed on all machines in a certain sequence, which may

be the same for all parts or different for different groups. The known time a_{ij} required to process the *j*-th part on the *i*-th machine (for those parts *j* that do not require processing on the *i*-th machine, $a_{ij} = 0$).

Manufacturing operations are considered integer, i.e., having started processing part j on machine i, it is necessary to complete it without interruption.

It is necessary to establish such a work order for each machine so that the total operating time of all parts is as minimal as possible. As noted above, the tasks of development and placement of production have received very significant application in the practice of planning and economic work.

These include the problems of determining the specialization and concentration of production, finding ways to organize and cooperate, analyzing the location of construction of enterprises and their reconstruction. Determining the schemes for transporting raw materials and products, etc. In view of the huge volume of capital investments behind each such practical task, even minor improvements in planned designs actually lead to a large national economic effect. In the problems of development and placement, the transport factor plays an important role. Many models relate to production-transport problems in an integer formulation.

The simplest model of placement is the so-called open transport problem, which consists of the following.

Let there be production points A_i, $i = 1,2,...,m$ of some homogeneous product with possible production volumes a_i and consumption points B_j, $j = 1,2,...,n$ with consumption volumes b_j .(Since we are talking about the production and consumption of some homogeneous product, we will identify production points with enterprises for the production of this product, which can be built.) In this case, the total volume of potential production of all suppliers exceeds the total volume of consumption, i.e.

$$\sum_{i=1}^{m} a_i > \sum_{j=1}^{n} b_j \qquad (14)$$

As in a normal transportation problem, the cost c_{ij} of transporting a unit of product from A_i to B_j is known. It is required to determine the supply plan $[x_{ij}]$, satisfying the conditions:

$$\sum_{j=1}^{n} x_{ij} \le a_i,\ i = \overline{1,m}; \tag{15}$$

$$\sum_{i=1}^{m} x_{ij} \le b_j,\ j = \overline{1,n}; \tag{16}$$

$$x_{ij} \ge 0 \tag{17}$$

and which minimizes total costs, i.e.

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij} \to \min.$$

This problem differs from the classical transport problem only in that instead of the usual balance of production and consumption we have inequality (14). Due to this, in conditions (15) instead of equalities $\sum_{j=1}^{n} x_{ij} = a_i$ there are inequalities.

The solution of problem (14)-(17) does not cause any difficulties, because with the help of a known method (introduction of a fictitious consumer) it is reduced to a transport problem. The optimal plan for this problem contains a list of real suppliers and their planned capacities, as well as a scheme for attaching suppliers to consumers.

Conditions (14)-(17) of the problem can be supplemented with restrictions, i.e. the volume of production at a given point must be lower than the specified one,

$$\sum_{j=1}^{n} x_{ij} \ge r_i.$$

When calculating the development and placement of production for the future using mathematical models, it is recommended to use some typical optimization problems. The latter are classified according to a number of features: problems for minimum costs, maximum national economic effect, maximum output; single-product and multi-product; static and dynamic with discrete and continuous variables; production and production-transport, etc. Depending on the set of features that appear when setting the problem, the latter

can be attributed to one or another type of optimization problem and described by the corresponding typical economic and mathematical model.

Let us give an example of a model of a production and transport multi-product static problem with discrete variables. The following notations are adopted:

i - index of the type of finished product or resource $(i = 1, 2, ..., n)$;

j - production facility index $(j = 1, 2, ..., m)$;

k - index of consumption point of products $(k = 1, 2, ..., l)$;

r - index of production method of the object $(r = 1, 2, ..., R_j)$;

a_{ij}^r - is the volume of output of product i $(i = 1, 2, ..., n^{\square})$ or the cost of resource type i $(i = n^{\square} + 1, ..., n)$ during production using method r at facility j;

d_{ki} - the given volume of consumption of product type i $(i = 1, 2, ..., n^{\square})$ at point k;

c_j^r - costs of production by method r at facility j;

p_i - the total volume of use of resource i $(i = n^{\square} + 1, ..., n)$ allocated to a given industry;

S_{jki} - the cost of transporting a unit of product i $(i = 1, 2, ..., n^{\square})$ from facility j to consumer k;

x_{jki} - volume of transportation of product i from facility j to consumer k;

z_j^r - the desired intensity of use of method r on object j.

The problem is reduced to finding such values of variables z_j^r and x_{jki} that minimize the value of the objective function

$$\sum_{j=1}^{m} \sum_{r=1}^{R} c_j^r z_j^r + \sum_{j=1}^{m} \sum_{k=1}^{l} \sum_{i=1}^{n^{\square}} s_{jki} x_{jki}$$

and the following conditions are met:

$$\sum_{r=1}^{R} a_{ij}^r z_j^r - \sum_{k=1}^{l} x_{jki} \geq 0 \quad (i = 1, 2, ..., n^{\square})$$

- the total volume of deliveries of product i by facility j to all consumers cannot exceed the corresponding volume of production;

$$\sum_{j=1}^{m} x_{jki} = d_{ki} \quad (i = 1, 2, ..., n^{\square})$$

- the given need of point k for the product of type i must be fully satisfied;

$$\sum_{j=1}^{m}\sum_{r=1}^{R} a_{ij}^{r} z_j^{r} p_i \le p_i \quad (i = n^{\square}+1,...,n)$$

- all objects of the industry can use resources of various types within the limits established for the industry;

$$\sum_{r=1}^{R_j} z_j^{r} \le 1;$$

$$z_j^{r} = 0 \vee 1$$

- the plan either includes only one production method at object *j* or does not include it at all (i.e. the object is not included in the plan);

$$x_{jki} \ge 0$$

- production volumes cannot be negative.

Single-product models are special cases of the corresponding multi-product models. The difference is that the number of types of resources or products included in the conditions of the problem is equal to 1, and thus the need for the index *k* disappears.

Methods for solving linear integer programming problems [15,24-27,118,134,138].

1. A number of special methods have been developed to solve integer programming problems

The simplest and most accurate of the methods for solving the integer problem (1)-(4) consists of solving the corresponding continuous problem (1)-(3) and then rounding the results to the nearest integers so that they also satisfy condition (4). The use of such a method is justified primarily in the case of mass production of hundreds and thousands of parts or some other products and the integer value can be neglected; rounding in this case will not lead to noticeable errors. For example, if when solving a specific integer programming problem, we obtain a solution (in a continuous statement), say *x = 1407.35* parts, then the result can easily be rounded to the integer *1407*.

However, this method is not always acceptable. For example, in the problem:

$$x_1 - 3x_2 + 3x_3 \to \max$$

under the following conditions

$$2x_1 + x_2 - x_3 \le 4;\ 4x_1 - 3x_2 \le 2;\ -3x_1 + 2x_2 + x_3 \le 3;$$

$$x_j \ge 0,\ j = 1,2,3,\ x_j - \text{ integers},$$

discarding the integer condition, we find the optimal solution to the corresponding continuous linear programming problem:

$$x_1 = 0,5,\ x_2 = 0,4,\ x_3 = 4,5.$$

It can be verified that no rounding options for the components of the non-integer solution here yield a feasible solution.

The optimal integer solution to this problem will be as follows:

$$x_1 = 2,\ x_2 = 2,\ x_3 = 5.$$

At the same time, the idea of discarding the condition of integerness and solving the corresponding continuous problem underlies a wide class of numerical methods of integer programming, called cutting methods.

2. The methods of cutting off are characterized by:

by temporarily discarding the conditions of the integer nature of the given problem;

by solving the resulting continuous problem (let's call it an auxiliary problem) using known optimization methods;

by checking the resulting optimal solution for optimality (if it exists) of the integer problem; if not, then by further transition to an integer solution [15,31,105,118].

In this case, an additional optimal constraint is constructed (it must be linear), with the help of which a part of the region is cut off, containing the optimal solution of the initial problem in a continuous statement and not containing any admissible solution of the integer problem. The process of constructing additional constraints and solving the resulting problems is the very essence of the cutting methods and continues until an optimal solution of the integer problem is obtained.

The cutting methods are based on the principles of multi-stage linear approximation of the problem and the transition from stage to stage using the so-called correct cutting; these principles can be implemented if a method for constructing correct cuttings is specified, ensuring the finiteness of the solution process.

Based on the general scheme of cutting methods, an algorithm for solving completely integer problems has been developed [links to cutting methods].

The algorithm for solving completely integer problems largely coincides with the general scheme of the cutting method:

1) The given problem is solved without the requirement of integerness;

2) The obtained optimal solution of this continuous problem, if it exists, is checked for integerness.

If the integer condition is satisfied for all variables, then the optimal solution of this problem is the optimal solution of the integer problem.

If the integer condition is not satisfied for at least one variable, then we proceed to the third stage.

If the continuous problem (1)-(3) turns out to be unsolvable, then the integer problem (1)-(4) also has no solution;

3) An additional constraint is constructed that has the property that it is used to cut off a part of the domain that contains the optimal solution to the continuous problem and does not contain a single admissible solution to the integer problem.

The process of constructing additional constraints and solving the resulting problems continues until we obtain an integer solution or are convinced that the problem is unsolvable.

An important problem of the cutting method is the increase in the number of additional constraints as auxiliary problems are solved, i.e. their dimension increases. There is a method that limits the size of the considered extended simplex tables of auxiliary problems. This method is based on the fact that additional constraints are of interest to us as a way of cutting off a non-integer optimal solution to an auxiliary problem and moving on to the next problem.

By introducing additional constraints, we obtain a new problem with $m+1$ equations. To the last simplex table we add another row corresponding to the additional constraint, and a column for the variable x_{n+1}. In this case, the relative estimates of c_j do not change, i.e. the optimality condition is preserved.

Continuing the process of transforming simplex tables, we apply the dual, simplex method and the variable x_{n+1} is derived from the basis. If after this we obtain a basic non-negative solution with integer components, then the problem is solved. If not, then we introduce new constraints with new variables and continue the dual simplex process until we obtain an optimal solution with integer non-negative values of the variables.

Another algorithm of the cutting method is intended for solving partially integer linear programming problems. Its computational scheme is also applicable to solving completely integer problems and differs only in the rule for constructing the coefficients of the correct cutting.

3. In addition to the cutting methods, which have been widely used until recently to solve integer programming problems, the branch and bound method is used to find optimal solutions to various optimization problems, integer

programming, and combinatorial optimization. The method is a development of the exhaustive search method, which, unlike the latter, eliminates subsets of admissible solutions that obviously do not contain optimal solutions.

The general idea of the method can be described using the example of finding the minimum of the function *f(x)* on the set of admissible values of the variable *x*. The function *f* and the variable x can be of arbitrary nature. The branch and bound method requires two procedures: branching and finding estimates (bounds) [14,51,79,82].

The branching procedure consists of dividing the set of admissible values of the variable *x* into subdomains (subsets) of smaller sizes. The procedure can be applied recursively to the subdomains. The resulting subdomains form a tree called a search tree or a branch-and-bound tree. The nodes of this tree are the constructed subdomains (subsets of the set of values of the variable *x*).

The procedure for finding estimates consists of finding upper and lower boundaries for solving the problem on a subdomain of admissible values of the variable *x*.

The branch and bound method is based on the following idea: if the lower bound of the function values on a subdomain *A* of the search tree is greater than the upper bound on any previously viewed subdomain *B*, then *A* can be excluded from further consideration (the elimination rule). Usually, the minimum of the obtained upper estimates is written to the global variable *m*; any node of the search tree whose lower bound is greater than the value of m can be excluded from further consideration.

If the lower bound for a tree node coincides with the upper bound, then this value is the minimum of the function and is achieved on the corresponding subdomain.

The method is used to solve some *NP*-complete problems, including the traveling salesman problem and the knapsack problem.

The following constructions are performed in the branch and bound method:

calculation of the lower bound (estimate);

division into subsets, i.e. branching;

recalculation of estimates;

finding solutions;

determination of the optimality criterion;

assessment of the accuracy of the approximate solution.

These constructions allow to significantly reduce the volume of solution enumeration.

To implement the branch and bound method as applied to individual discrete optimization problems, it is necessary, based on their features, to specify the rules of branching, calculation of estimates (boundaries) and finding solutions.

Based on the methods of cutting and the branch and bound method, hybrid methods are developed, the development of which proceeds mainly along the path of creating new numerical methods for solving both general problems of discrete optimization and problems of a particular type.

4. Combinatorial methods for solving combinatorial problems (knapsack problems, traveling salesman problems, assignment problems, etc.) are being intensively developed.

A combinatorial problem is defined as follows. Let there be a set of n elements. On this set, a set of combinations $P = \{P_1, P_2, ..., P_n\}$, is defined, where combinations are understood as combinations, substitutions, and rearrangements inherent in each problem. A function f is defined on the set P. In a combinatorial problem, the extremum of f (maximum or minimum) is sought, and those elements of the set P on which the function f is extremal are found. When solving combinatorial problems, one must be able to sort through a set of values of the function f, and then calculate these values and compare them.

A large number of algorithms have been developed to solve combinatorial problems, the general idea of which is to replace the complete enumeration of all variants with partial enumerations of smaller volumes. The implementation of such an algorithm is carried out by discarding some subsets that obviously do not contain the desired extremum and narrowing the area of promising variants. In this case, very diverse methods are obtained, determined by the structure of the corresponding finite sets P.

5. At present, approximate methods, heuristic methods, local optimization methods, random search methods, in particular methods based on ideas borrowed from nature (genetic, ant colonies, etc.) are used to solve discrete optimization problems.

The enumeration methods and all their improvements require running time, which usually grows exponentially with increasing problem dimension, which is unacceptable for solving practical problems. Research in the field of discrete optimization methods has made it possible to find a number of

problems for which effective (i.e. polynomial with a low degree of the polynomial) solution algorithms do exist.

These include, for example, such problems as finding the shortest path between graph nodes, maximizing the flow in a network for a given edge throughput, etc. However, for most problems that arise in practice, effective algorithms for an exact solution are unknown. Therefore, approximate and heuristic methods are used to solve problems.

In them, instead of strict proof that the resulting plan will give the exact minimum or maximum of the objective function, one has to rely on common sense considerations, which allow one to assume that the resulting plan is, with a sufficiently high probability, either optimal or at least close to optimal.

Optimization algorithms for which there are non-trivial estimates of the possible deviation of the solution from the optimum are called approximate or suboptimal. A good example of an approximate algorithm is known for the problem of multiprocessor scheduling.

Given *m* identical processors and *n* independent tasks, each of which can be solved on any processor. The time to solve each task is t_i, $i = \overline{1,n}.$. How to distribute tasks among processors so that all tasks are completed in the shortest possible time?

The algorithm consists of arranging the tasks in descending order of their duration, and then assigning each subsequent task to the least loaded processor. It is quite typical that the algorithm used gives quite decent solutions, but there is no guarantee that these solutions are always close to optimal. This is an example of the application of a heuristic algorithm, very widely used in discrete optimization.

Algorithms based on heuristic (plausible) considerations are called heuristic algorithms. There are no universal recipes for constructing heuristic algorithms for any task. At the same time, there are several well-proven approaches that prove useful for different types of tasks.

Local step-by-step optimization algorithms are based on the following considerations. In most combinatorial problems, the construction of a plan can be organized as a step-by-step process. For example, at each step of the solution, the value of the next variable can be determined. In most cases, it is difficult to say what decision should be made at the next step in order to eventually arrive at the optimal plan for the problem. However, it is often easy to choose the solution that seems best for a given step, without taking into account the long-term consequences.

The simplest type of local step-by-step optimization algorithms are sequential choice algorithms.

Let's look at the traveling salesman problem as an example. We randomly select a starting city and ask ourselves which city we should go to first. The obvious answer is the closest one. At the second step and at subsequent steps, using the same logic, we should select the closest of the cities that we have not yet visited. Of course, such an algorithm does not guarantee the optimality of the constructed plan, since it is possible that winning at the first steps will lead to greater losses at subsequent ones. However, we can confirm by probabilistic reasoning that when choosing the closest city at each step, the expected length of the entire route will be less than with any other choice.

For the knapsack problem, we can propose the following sequential selection algorithm: sort the goods in descending order of the ratio of the cost of a unit of goods to its volume, and then at each step take the maximum amount of the next goods that fits in the remaining volume.

The performance of sequential choice algorithms can be improved by allowing a limited search, i.e. choosing a solution at a given step, taking into account the results of two, three or more steps. For example, by searching through all the options for the first two trips of a traveling salesman, choosing the option that has the minimum length, and making the first step based on this choice.

Then, in the same way, try the options for the second and third steps, take the second step, etc. If the number of steps to be tried is limited, the algorithm, although it will work much slower, will still retain polynomial complexity. Of course, due to the limited depth of the search, obtaining an optimal plan is not guaranteed. It is not even guaranteed that with an increase in the depth of the search, the quality of the plans obtained will always increase.

Another type of local step-by-step optimization algorithms are sequential plan improvement algorithms. Let us assume that some (non-optimal) plan for a problem has already been constructed in one way or another (for example, using a sequential choice algorithm). Sometimes it is possible to propose a procedure for a simple modification of the plan, which will leave it as a feasible plan for the problem.

If the value of the objective function is better for the modified plan than for the original one, then the new, improved plan should be taken instead of the original one and an attempt should be made to apply the improving modification to it again. The process stops when all possible modifications can only worsen the plan.

The main disadvantage of local optimization algorithms is that they do not necessarily lead to a globally optimal plan for the problem. In particular, for successive improvement algorithms, a good choice of the initial plan is very important. Using random selection to construct the initial plan can play a useful role.

Random search. In random search, all combinations are selected randomly according to some distribution law. The values of the objective function on these combinations are calculated and the combination that gives the extreme value of the objective function is selected. The main problem in random search is the selection of essential features of combinations. There are two approaches: a) features are selected in advance by analyzing the conditions of the problem; b) features are obtained from the analysis of already obtained solutions. A randomly selected plan will most likely be worse in itself than a plan constructed using the sequential selection algorithm (although not necessarily), but with multiple repetitions of the combination "random selection + sequential improvement" the probability of arriving at a global optimum increases.

CHAPTER 1. ON ONE CLASS OF PROBLEMS WITH SEPARATING SETS OF CONSTRAINTS

1.1 On the definition and solution of problems with separating sets of constraints

An analysis of many applied problems formulated as integer linear programming or Boolean programming problems shows that they have the following features: their constraints are divided into two sets, which we denote by *A* and *B*. For the constraints of set *A*, it is possible (relatively simple) to develop an effective (fast) generator of vectors satisfying them. Another group of constraints, which make up set *B*, is such that the admissibility check of the vector obtained by this generator is carried out fairly quickly due to the use of the specifics of these constraints [28-62].

Let us formulate several types of restrictions with the specified property:

a)

$$\sum_{k=1}^{l}\sum_{j=1}^{m} u_{kl}\delta_{kj} \le c$$

$$\sum_{k=1}^{l} \delta_{kl} \le 1,\ j = \overline{1,m}$$

$$\delta_{kl} \in \{0,1\}, k = \overline{1,l};\ j = \overline{1,m}$$

b)

$$x_j + y_j - y_{j+1} = d_j$$

$$x_n + y_n = d_n$$

$$x_1, x_j, y_j - \text{ integers},\ j = \overline{2,n}$$

c)

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, i = \overline{1,m}$$

$$x_j \ge 0,\ \text{integers},\ j = \overline{1,n}$$

moreover $b_i > 0, a_{ij} > 0,\ i = \overline{1,m};\ j = \overline{1,n}$

d)

$$\sum_{i=1}^{m} x_{ij} = 1,\ j = \overline{1,n}$$

$$\sum_{j=1}^{n} A_{ij} x_{ij} \le B_{i,}\ i = \overline{1,m}$$

$$x_{ij} \in \{0,1\}, i = \overline{1,m};\ j = \overline{1,n}.$$

For example, for the group of constraints from d)

$$\sum_{i=1}^{m} x_{ij} = 1,\ j = \overline{1,n}$$

$$x_{ij} \in \{0,1\}, i = \overline{1,m};\ j = \overline{1,n}.,$$

which we assign to the set *A*, it is easy to develop a generator of vectors satisfying them. The remaining constraints from d) form the set *B* and have a feature that is well visible if we represent the constraints d) in the form $Gx \le b,$, where *x* is an $m \cdot n$-dimensional vector $x = (x_{11}, ..., x_{m1,} x_{12}, ..., x_{m2}, x_{1n}, ..., x_{mn})$.

If the coefficients A_{ij} in constraints d) were non-negative, then all constraints could be assigned to set *A*, and set *B* would be empty. Thus, the division of the set of constraints into two groups is ambiguous and is carried out in practice depending on the complexity of the generator of vectors satisfying constraints *A*.

The class of problems with separable sets of constraints, for example those defined in a)-d), also represents, accordingly, applied problems from various areas of the national economy: the problem of determining the development strategy of a technical object, the problem of planning the release of specialists; the frequently encountered multidimensional knapsack problem with positive coefficients.

The list of problems with separable sets of constraints could be continued. However, it is already clear from the above problems that this class is quite meaningful. For it, as in general, for the class of discrete programming problems, the methods known in discrete optimization are applicable: cutting, branches and boundaries, sequential analysis of options, construction of a sequence of solutions, random search with local optimization, etc.

There are known difficulties that one has to face when using some integer linear programming methods: 1) slow convergence rate, 2) accumulation of rounding errors at intermediate iterations, 3) absence (in approximate algorithms) of a criterion for the optimality of the found solution, 4) the need

for a large amount of memory to bring the solution to the optimal one and, in general, insufficient efficiency in solving problems of sufficiently large dimensions.

The available information on solving large-scale problems (of the order of 1000 or more variables and 100 or more constraints) relates, as a rule, to problems of a special type, to random phenomena. In addition, the accumulated experience of solving discrete optimization problems shows that in this area there is no effective algorithm for solving all problems, including those that arise in practice. We can say that for two given algorithms *A* and *B*, as a rule, there will be a problem that is solved by algorithm *A* faster than by algorithm *B*. Just as there will be another problem that can be solved by algorithm *B* faster than by algorithm *A*. For these reasons, the issue of developing new or modifying known exact or approximate algorithms that expand the possibilities of solving integer linear programming problems, in particular, problems with separable sets of constraints, is relevant.

1.2. Modification of the cutting method

The question of the prospects of exact methods for solving integer linear programming problems, in particular, the cutting method, is debatable. It is known that they are of little use in practice. Nevertheless, they are being developed and improved. For their study, the apparatus of other mathematical disciplines is used (group theory, dynamic programming, nonlinear programming, etc.). The cutting method has not yet been sufficiently studied. Therefore, modifications of this method are proposed, which are applicable to problems with separable sets of constraints as a special case of integer linear programming problems. Apparently, in the future, new and well-known numerical (computational) schemes of the cutting method will be developed and improved with the aim of its more complete study and application.

Let it be required to solve a problem of type

$$\max \sum_{i=1}^{n} c_j x_j \tag{1.1}$$

under constraints

$$\sum_{o=1}^{m} a_{ij} x_j = b_i,\ i = \overline{1,m} \tag{1.2}$$

$$x_j \geq 0,\ j = \overline{1,n} \tag{1.3}$$

$$x_j \text{ - integer, } j=\overline{1,n}, \tag{1.4}$$

with integer coefficients $a_{ij}, b_i, c_j,\ i=\overline{1,m},\ j=\overline{1,n}.$

We will solve this problem using the first Gomory algorithm. (Note that Hadley [93] writes about the high competitiveness of the first Gomory algorithm compared to the third Gomory algorithm or the Land and Doig method).

After applying the simplex method to solve the continuous problem (1.1)-(1.3), the original problem can be represented in the form

$$\max(\mathrm{f}+\sum_{j\in N} z_j(-x_j)) \tag{1.5}$$

under constraints

$$x_i = xp_i + \sum_{j\in N} x_{ij}(-x_j),\ i\in M, \tag{1.6}$$

$$x_i \geq 0, x_j \geq 0,\ i\in M,\ j\in N, \tag{1.7}$$

$$x_i, x_j - \text{ integers, } i\in M,\ j\in N, \quad . \tag{1.8}$$

where M is the set of indices i of the basic variables x_i, N is the set of indices j of the non-basic variables x_j,

If is the value of the objective function (1.1) on the set of continuous solutions. If the objective function takes a non-integer value on the optimal solution from the set (1.6)-(1.8), as well as for a non-integer solution *xp*, we add the constraint to the constraints (1.9)

$$s \geq 0,\ s \text{ - integer} \tag{1.9}$$

where

$$s = \begin{cases} -\{\mathrm{f}\} - \sum_{j\in N} z_j(-x_j)),\ if\ \{\mathrm{f}\}>0, \\ -\{xp_i\} - \sum_{j\in N} \{x_{ij}\}(-x_j),\ if\ \{\mathrm{f}\}>0,\ \{xp_i\}>0. \end{cases}$$

We treat the resulting problem in the same way as the original one

until we obtain an integer non-negative solution. After introducing the cutting (1.9), the value of the objective function does not necessarily change. On such inefficient (definition in [15]) the possibility of forming the cutting using the representation (1.5) of the objective function is not used, since the value of the objective function is an integer (otherwise the objective function would change its value).

To speed up the convergence of the cutting method, it is also possible, in the case where the value of the objective function is an integer, to use representation (1.5) to form an additional constraint. Indeed, at some iteration of the cutting method, the value of the objective function is equal to f, the values of the dual estimates are equal to $z_{j,}\ j=\overline{1,n}$, and in the basic solution the *i*-th component xp_i is non-integer. There can be two cases depending on the value f' of the objective function after introducing the cutting (1.9) at the next iteration: a) $f' < f$ or b) $f' = f$.

In case a) the iteration performed leads to a decrease in the value of the objective function. However, quite often, especially after a certain number of iterations, case b) occurs and is repeated at subsequent iterations, i.e. the value of the objective function does not decrease at subsequent iterations. In cases b) it is proposed to use a cutting different from (1.9). This cutting has the form

$$\sum_{j\notin Z}\{x_{ij}\}x_j \geq \{xp_i\}, \qquad (1.10)$$

where $Z=\{j \mid j\in N, z_j \neq 0\}$ and i such that $\{xp_i\}>0$.

It is useful to introduce the cutoff (1.10) in order to quickly answer the question about the possibility of changing the objective function of its value f. From here, the main result can be formulated as follows:

Lemma 1.1. If the new problem obtained from problem (1.5)-(1.8), (1.10) after equating the variables x_j with numbers x_j to zero turns out to be unsolvable, then an integer non-negative solution to problem (1.5)-(1.8) should be sought at a level of the objective function that is smaller in comparison with the previously obtained value of f.

Using this result, we can describe a new algorithm 1.1 for the completely integer problem (1.1)-(1.4) with integer coefficients $a_{ij}, b_i, c_j, i=\overline{1,m;}\ j=\overline{1,n,}$, which is a variant of the first Gomory algorithm.

Algorithm 1.1.

Step 1. The problem (1.1)-(1.3) is solved by the simplex method and the representation (1.5)-(1.8) is obtained; if xp is an integer solution, then the counting stops.

Step 2. If f is a non-integer number, then the counting continues according to the first Gomory algorithm.

Step 3. If f is an integer, then a new problem is formed from the problem obtained in the last step without taking into account the columns with numbers j and the dual estimate z_j that is not equal to zero.

Step 4. The new problem is solved using the first Gomory algorithm.

Step 5. If the optimal solution to the new problem is integer, then it will be the same for the original problem and the count ends.

Step 6. If the new problem turns out to be inconsistent, then a constraint is introduced in problem (1.1)-(1.4)

$$\sum_{j \in N} z_j x_j \geq 1$$

and the calculations begin with step 1. To do this, it is necessary to restore the problem from which the new one was composed.

Algorithm 1.1 is fully described. Note the following property of the cutting (1.10).

Theorem 1.1. The cutting (1.10) cuts off the non-integer optimal solution of problem (1.5)–(1.8), but does not cut off any non-negative integer solution at the level of the objective function (1.5) with value f.

Proof. Consider the level of the objective function with the value f. All feasible solutions (including non-negative integer ones) lying on it satisfy the relation

$$\sum_{j \in N} z_j x_j = 0.$$

Since $z_j \geq 0$, then for those j for which $z_j > 0$, we have $x_j = 0$. Thus, in order for problem (1.5)–(1.8) to have a non-negative integer solution with the value f of the objective function, it is necessary that the problem obtained by deleting columns with numbers j for which $z_j > 0$ also had the same solution. Cutting (1.10) is the Gomory cutting, but for this new problem, so it does not cut off integer non-negative solutions. Theorem 1.1 is proved.

Theorem 1.2. The finiteness of the described algorithm 1.1 follows from the finiteness of the first Gomory algorithm, the conditions for the convergence of which are assumed to be satisfied.

Remark 1.1. Algorithm 1.1 can be included in the scheme of the first Gomory algorithm after the last few iterations have been completed.

Remark 1.2. Similar reasoning can be made regarding the third Gomory algorithm [15]. It is obvious that Algorithm 1.1 is useful for problems in which

the objective function has many nonzero dual estimates. (The number of such estimates changes from iteration to iteration.) In this case, the dimension (in terms of the number of variables) of the new problem is sharply reduced compared to the generating one.

Algorithm 1.2 is based on a generalization of algorithm 1.1, which combines the usual Gomory cutting with cutting at a fixed level of the objective function. The usefulness of considering such an algorithm is explained as follows. The cutting algorithm, as a rule, is accompanied by a monotonic decrease in the objective function to a certain value , then becomes ineffective, i.e. the objective function does not change over a large number of subsequent iterations.

In such cases, it is natural to consider the level corresponding to the last obtained integer value of the objective function, and to look for an integer solution on it. In the absence of an integer solution on this level, we decrease the value of the objective function by some positive number (per unit, for example, if its optimal value is an integer). Next, we look for a non-negative integer solution at a new level.

A concrete implementation of this approach is presented here by Algorithm 1.2, which relies heavily on Theorem 1.1. It appears to be more general in the sense that the idea of Algorithm 1.1, first presented in [31], is repeatedly applied.

Let the coefficients a_{ij}, b_i, c_j in problem (1.1)-(1.4) be integers and the region of feasible solutions be limited. In what follows, we consider precisely this problem, which is denoted by *T*. We present a formalized description of algorithm 1.2.

Algorithm 1.2.

Step 1. Solve the continuous problem T_0 corresponding to the integer problem T. If T_0 is inconsistent, then the calculation is complete. Otherwise, we obtain a simplex table

$$CT_0 = \begin{pmatrix} f^0 & z_1^0 & \dots & z_{n^0}^0 \\ xp_1^0 & x_{11}^0 & \dots & x_{1n^0}^0 \\ \dots & \dots & \dots & \dots \\ xp_{m^0}^0 & x_{m^0 1}^0 & \dots & x_{m^0 n^0}^0 \end{pmatrix}$$

Step 2. If the solution f^0 of the objective function is non-integer, then the calculation continues according to the first Gomory algorithm.

Step 3. If f^0 is an integer and the corresponding solution is integerness, then the calculation ends, since the optimal integer solution has been found.

Step 4. If f^0 is an integer and the corresponding solution is non-integerness, then problem T_i is composed with $i=1$: for all those j for which $z_j^0 \neq 0$, in problem T_0 the non-basic variable x_j is equated to zero. With respect to all remaining non-basic variables (if any), the set of indices of which we denote by N_1, the condition of integerness is preserved. Under these restrictions, problem T_i consists of maximizing

$$\bar{x}_1 = xp_1^0 + \sum_{j \in N_1} x_{1j}^0 (-x_j).$$

Step 5. Solve the continuous problem T_1^c, corresponding to problem T_1. If the conditions of the continuous problem are contradictory, then the calculations are performed from step 13. Otherwise, a simplex table is obtained

$$CT_i = \begin{pmatrix} f^i & z_1^i & \dots & z_{n^i}^i \\ xp_1^i & x_{11}^i & \dots & x_{1ni^1}^i \\ \dots & \dots & \dots & \dots \\ xp_{m^i}^i & x_{m^i 1}^i & \dots & x_{m^i n^i}^i \end{pmatrix}$$

Step 6. If xp_1^1 is not an integer, then the process of solving problem T_1 continues using the first Gomory algorithm.

Step 7. If xp_1^1 is an integer and all xp_i^1, $i = \overline{2, m^1}$ are integers, then the optimal solution to problem T_1 is determined, and therefore the calculation for problem *T* is complete.

Step 8. Problem T_{i+1} is constructed from problem T_i. If the continuous problem corresponding to problem T_i is solved and its optimal simplex table CT_i has the form

$$CT_i = \begin{pmatrix} f^i & z_1^i & \dots & z_{n^i}^i \\ xp_1^i & x_{11}^i & \dots & x_{1ni^1}^i \\ \dots & \dots & \dots & \dots \\ xp_{m^i}^i & x_{m^i 1}^i & \dots & x_{m^i n^i}^i \end{pmatrix}$$

then problem T_{i+1} is constructed as follows: the set z^i of indices $z^i = \{j \mid j \in \{1,...,n^i\}, x_{ij}^i \neq 0\}$ is considered and in problem T_i the non-basic variable x_j is set equal to zero if $j \in z^i$. For the remaining non-basic variables (the set of their indices is denoted by N_{i+1}), the integer condition is preserved. Under these constraints, problem T_{i+1} consists of maximizing

$$\bar{x}_{i+1} = xp_{i+1}^i + \sum_{j \in N_{i+1}} x_{i+1,j}^i (-x_j).$$

Step 9. Solve problem T_{i+1}^c (a continuous problem corresponding to problem T_{i+1}). If T_{i+1}^c is inconsistent, then calculations are performed starting from step 14.

Step 10. If xp_i^i is not an integer, then the process of solving problem T_{i+1} continues according to the first Gomory algorithm.

Step 11. If xp_i^i is an integer and $xp_s^i, s = i+2,...,m^i,$ is an integer, it means that the optimal solution to problem T has been found and the calculation ends.

Step 12. If xp_i^i is an integer and there is a non-integer among $xp_s^i, s = i+2,...,m^i, -$, then i is increased by one and the calculations from step 8 are performed.

Step 13. The last task with number T_0 is considered. If $f^0 - 1 \geq f_{\min}$ (the minimum value of the objective function of task T_0), then an additional constraint is introduced

$$\sum_{j \in N_0} z_j^0 x_j \geq 1.$$

A new problem T_0 is obtained and calculations are performed from step 1.

Step 14. Return to problem T_i^c. If the feasible region allows a decrease in $\bar{x}_i$ by one, then an additional constraint is introduced in it

$$\sum_{j \in N_i} x_{ij} x_j \geq 1.$$

A new problem T_i is obtained and calculations begin with step 5 if $i = 1$, and with step 9 if $i > 1$.

Step 15. If $\bar{x}_i$ cannot be decreased by one because it is outside the feasible solution region, then *i* is decreased by one and the calculations begin with step 14.

Algorithm 1.2 is fully described. Its efficiency depends significantly on the dimensions of the problems T_i obtained in the process of computing, in other words, on the number of non-zero dual estimates of the auxiliary problems T_i. Therefore, it is recommended to use Algorithm 1.2 for solving problems with densely filled simplex tables, in which there are many non-zero dual estimates. The following theorems 1.3 and 1.4 relate to Algorithm 1.2.

Theorem 1.3. $n^1 < n^0$ is always satisfied, i.e. the dimension of problem T_1 is less than the dimension of problem T_0.

Proof. Let there be *p* elements in a set $z^0 = \{j \mid z_j^0 = 0\}$, where z_j^0 is from table T_0. If $p = 0$, then the optimal value of problem T is known. Then we can introduce a constraint $cx^0 = f$, in the latter, thus eliminating one variable in problem T. Now in problem T_1 the number of variables will be at least one unit less than the number of variables in problem T_0. If *p>0*, then since $n^1 = n^0 - p$, then in this case $n^1 < n^0$.

Remark 1.3. A theorem similar to 1.3 is valid for each pair of problems T_1 and T_{i+1}, i.e. we can assume that $n^{i+1} < n^i$.

Remark 1.4. To control the calculations, we note that the simplex table CT_i corresponding to problem T_1 is represented as follows:

$$CT_i = \begin{pmatrix} f^i & 0 & \dots & 0 \\ xp_1^i & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ xp_{i-1}^i & 0 & \dots & 0 \\ xp_i^i & x_{i1}^i & \dots & x_{in^i}^i \\ \dots & \dots & \dots & \dots \\ xp_{m^i}^i & x_{m^i 1}^i & \dots & x_{m^i n^i}^i \end{pmatrix}$$

where f^i is an integer and xp_k^i, $k = \overline{1, i-1}$, are non-negative integers.

Theorem 1.4. Algorithm 1.2 finds an optimal solution to a problem with a bounded polyhedron of feasible solutions or establishes the absence of such a solution in a finite number of operations.

The proof follows from the construction of algorithm 1.2 and from the finiteness of the first Gomory algorithm, which takes place under the given assumptions and using the dual lexicographic simplex method.

Remark 1.5. With some modifications, Algorithm 1.2 can be extended to problems of mixed integer linear programming.

Indeed, let us consider problem (1.1) - (1.4) , in which the condition of integerness of all variables is replaced by condition

$$x_j \equiv 0(\mathrm{mod}1),\ j \in K \subset \{1,...,n\} \qquad (1.11)$$

and assume that the optimal value of the objective function (1.1) under constraints (1.2) - (1.3), (1.11) is an integer. Then, for the applicability of algorithm 1.2, it is necessary:

1) solve the corresponding continuous problem,

2) arrange and fix the variables in the last simplex table so that any variable that must be an integer has a number smaller than any variable without the requirement of integerness,

3) perform the input of the cutoff according to the rule of the second Gomory algorithm [15],

4) perform the check for integerness of the basic variables at the 3rd, 7th and 11th steps according to (1.11).

Computational experiments show that for some problems in Algorithm 1.2 there is a decrease in both the number of iterations and the size of the generated problems, compared to the cyclic Gomory algorithm. Such situations arise, as a rule, in the initial period of the calculation, when the objective function monotonically decreases, and then it stabilizes. Then the question arises whether the objective function can have a smaller value. To answer the question, it is proposed to introduce new restrictions using the values of dual estimates of auxiliary problems.

1.3. Hermite's algorithm for solving systems of linear equations in integers

The Hermite algorithm for solving systems of linear equations

$$\sum_{j=1}^{s} \alpha_{lj} \mu_j = \beta_l,\ l = \overline{s+1, m}. \tag{1.12}$$

in integers plays an important role in integer linear programming (similar to Gaussian elimination in linear programming [13,29]. Its formalized presentation contains steps 0-13, as a result of which the following arrays of the corresponding dimensions are transformed:

$$A = \{\alpha_{ij}\},\ i = \overline{1, m};\ j = \overline{1, \mathrm{n}}$$
$$v = \{\mathrm{v}_{kj}\}, k = \overline{1, \mathrm{n}};\ j = \overline{1, \mathrm{n}}$$
$$u = \{u_i\},\ b = \{\beta_i\},\ \mu = \{\mu_i\},\ i = \overline{1, m}$$

Step 0. Initial formation of the specified arrays

$$\alpha_{ij} = a_{ij},\ v_{kj} = \delta_{kj},\ u_i = i,\ \beta_i = b_i,$$
$$p = \min(m, n),\ s = 1,$$

where a_{ij}, b_i is the given integer coefficients of the system (1.12), δ_{kj} is the Kronecker delta.

Step 1. Determine r and k such that

$$|\alpha_{rk}| = \min_{\alpha_{ij} \neq 0} |\alpha_{ij}|,\ i = \overline{s, m};\ j = \overline{s, n}$$

If $\alpha_{ij} == 0,\ i = \overline{s, m};\ j = \overline{s, n},$, then a check is made for the solvability of system (1.12). Namely, if *s=1*, then we check whether all β_i are equal to zero, $i = \overline{1, m}.$ If yes, then any integer vector $x = (x_1, ..., x_n)$ is its solution; if not, then the system is inconsistent. In both cases, the process ends. If *s=p* and *p<m*, then we check whether the conditions are met

$$\sum_{j=1}^{s} \alpha_{lj} \mu_j = \beta_l,\ l = \overline{s+1, m}. \tag{1.13}$$

If not, then system (1.12) is inconsistent and the calculation ends. If conditions (1.13) are met, then go to step 13.

Step 2. If *s=r*, then go to step 4.

Step 3. Permutation of the rows of matrix A with numbers *s* and *r* of the corresponding components of vectors *u* and *b*:

$$t = \alpha_{rj},\ \alpha_{rj} = \alpha_{sj},\ \alpha_{sj} = t$$

$$t = u_r,\ u_r = u_s,\ u_s = t$$

$$t = \beta_r,\ \beta_r = \beta_s,\ \beta_s = t,\ j = \overline{1,n}$$

Step 4. If $s = k$, then go to step 6.

Step 5. Permutation of columns of matrix A and v with numbers *s* and *k*:

$$t = \alpha_{ik},\ \alpha_{ik} = \alpha_{is},\ \alpha_{is} = t,\ i = \overline{1,m},$$

$$t = v_{jk},\ v_{jk} = v_{js},\ v_{js} = t,\ t = \overline{1,n}.$$

Step 6. If $s = n,$, then go to step 9.

Step 7. Recalculation of columns with $j = \overline{s+1,n}$ numbers of matrices A and v:

$$t_j = -[\alpha_{sj} / \alpha_{ss}],\ \alpha_{ij} = \alpha_{ij} + t_j \alpha_{is},\ i = \overline{s,m},$$

$$v_{kj} = v_{kj} + t_j v_{ks},\ k = \overline{1,\mathrm{n}}.$$

Step 8. If $\alpha_{sj} \neq 0$ for some $j = \overline{s+1,n}$, then go to step 1.

Step 9. Definition μ_s:

$$\mu_s = \begin{cases} \beta_s / \alpha_{ss},\ \mathrm{s} = 1 \\ \left(\beta_s - \sum_{j=1}^{s-1} \alpha_{sj} \mu_j \right) / \alpha_{ss},\ s \neq 1. \end{cases}$$

Step 10. If μ_s is an integer, then go to step 12.

Step 11. If μ_s is not an integer, then system (1.12) has no integer solutions and the count ends.

Step 12. If $p \geq s+1,$ then increase *s* by one and go to step 1.

Step 13. Obtaining all integer solutions $x = (x_1, ..., x_n)$ of system (1,12) with arbitrary integer parameters y_l :

$$\mathrm{x}_{j_0} = \sum_{i=1}^{s} v_{ji} \mu_i,\ j = \overline{1,n}$$

$$x_j = \begin{cases} x_{j_0},\ s = n \\ x_{j_0} + \sum_{j=s+1}^{n} v_{jl} y_l,\ s < n \end{cases} \qquad (1.14)$$

The Hermite algorithm is described. It follows directly from it

Corollary 1.1. Whatever the integer matrix $A=\{a_{ij}\}$, $i=\overline{1,m}$; $j=\overline{1,n}$ of rank *m*, there exist two unimodular matrices *u* and *v* such that

$$H=uAv=L+O, \tag{1.15}$$

where *O* is an upper triangular matrix of zero elements, and the matrix *L* has the form of a lower triangular matrix.

Reducing the matrix A to the form (1.15) is ambiguous. Indeed, taking, for example, $m=1$, $n=3$, $A=(3,5,7)$ for any integers *t* and *s* and unimodular matrices $u=(1)$ and

$$v=\begin{pmatrix} 5t-8 & -5 & 5s+1 \\ -3t+5 & 3 & -3s-2 \\ 0 & 0 & 1 \end{pmatrix}$$

we have $uAv=(1,0,0)$.

From (1.15) it follows that the question of the solvability of a system of linear equations in integers is reduced to the presence of solutions of some triangular system, also with an integer matrix. In particular, such a triangular system is the system $Ly=ub$.

When solving system $Ax=b$ in integers using the Hermite algorithm, at the *k*-th iteration two unimodular matrices u_k and v_k are constructed such that

$$H_0=A,\ H_k=u_kH_{k-1}v_k,\ k=\overline{1,r}.$$

Then u,v and H are determined

$$u=u_ru_{r-1}...u_1,\ v=v_1v_2...v_r,\ H=H_r=uAv.$$

Since u is a permutation matrix, the actual receipt of an integer solution depends on the growth of the modules of the elements of the matrix H_k at intermediate iterations of *k*. The study of this issue is associated with the introduced function $f(m,n,M)$.

Let $M=\max\limits_{i,j}|a_{ij}|$ be the maximum absolute value element of matrix A, h_{ij}^k be the element of matrix H_k, $i=\overline{1,\mathrm{m}}$; $j=\overline{1,\mathrm{n}}$; $k=\overline{1,r}$, and let function $f(m,n,M)$ be defined as follows:

$$f(m,n,M)=\max|h_{ij}^k|,\ i=\overline{1,\mathrm{m}};\ j=\overline{1,\mathrm{n}};k=\overline{1,r}$$

Theorem 1.5. $f(m,n,M)$ is a non-decreasing function with respect to m,n,M.

Proof. Let us fix m. For $m'>m$, we represent the matrix $(m'-m)n$ as $C=\begin{pmatrix}A\\B\end{pmatrix}$,

where the matrix B written below is an integer matrix of size $(m'-m)n$ with elements b_{ij} such that $|b_{ij}|\le M$. If B is a zero matrix, then applying the Hermite algorithm to the matrix C yields $f(m',n,M)=f(m,n,M)$. Therefore, for $B\neq 0$, the function $f(m',n,M)$ can only increase. The non-decrease of $f(m,n,M)$ with respect to n and M is proved similarly.

Theorem 1.6. For $m\ge 1$, the inequality holds

$$f(m,n,M)\ge M(M+1)^{m-1}.$$

Proof: Apply Hermite's algorithm to the matrix

$$C=\begin{pmatrix}1 & 0 & \dots & 0 & M\\ -M & 1 & \dots & 0 & M\\ \dots & \dots & \dots & \dots & \dots\\ -M & -M & \dots & 1 & M\\ -M & -M & \dots & -M & M\end{pmatrix}$$

of order $m*m$. The element $h^m_{m,m}$ of the matrix H, corresponding to the matrix C, will be equal to $M(M+1)^{m-1}$.

The estimate from Theorem 1.6 is not exact. In fact, the growth of the modules of intermediate elements is even higher. This follows from applying the Hermite algorithm to the matrix

$$A=\begin{pmatrix}F_n & F_{n-1}\\ F_{n+1} & -F_{n+1}\end{pmatrix},$$

where F_n is the *n*-th Fibonacci number.

The Hermite algorithm generates matrices

$$H_k=\begin{pmatrix}F_{n-k+1} & F_{n-k+2}\\ (-1)^{k+1}F_{k+1}F_{n+1} & (-1)^k F_kF_{n+2}\end{pmatrix}$$

at intermediate iterations k and finally leads to the matrix

$$H = \begin{pmatrix} 0 & 1 \\ (-1)^{n+2} F_{n+2}F_{n+1} & (-1)^{n+1} F_{n+1}^2 \end{pmatrix}.$$

From this it is clear that the largest absolute value number in the last matrix is equal to $F_{n+2}F_{n+1}$ and therefore greater than $F_{n+1}(F_{n+1}+1)$, $n \geq 1$.

Theorem 1.7. Let A be a matrix of size $m*n$ with integer elements a_{ij}. The maximum number of iterations *r* for obtaining the matrix *H* in cases $m \leq 2$ and $n = 2$ is determined from condition $M = \min_p F_p = F_{r-1}$.

Proof. In case $m = 1$, the maximum number of iterations in the Hermite algorithm is equal to the number of transitions from matrix (F_n, F_{n+1}) to matrix $H = (1,0),,$ and in case $m = 2$, it is equal to the number of transitions from matrix

$$\begin{pmatrix} F_n & F_{n-1} \\ F_{n+1} & -F_{n+1} \end{pmatrix}$$

to the matrix

$$\begin{pmatrix} 0 & 1 \\ (-1)^{n+2} F_{n+2}F_{n+1} & (-1)^{n+1} F_{n+1}^2 \end{pmatrix}$$

In both cases, the number of iterations is the same and equal to $n+2$.

Theorem 1.8. Let $m < n$. Representation (1.15) be invariant under right-hand multiplication by a matrix R,

$$R = \begin{pmatrix} 1 & 0 & ... & 0 & 0 \\ r_{21} & 1 & ... & 0 & 0 \\ ... & ... & ... & ... & ... \\ r_{n1} & r_{n2} & ... & r_{n,n-1} & 1 \end{pmatrix},$$

where r_{ij} are arbitrary integers.

Proof. Since the determinant of the matrix *r* is equal to one, the matrix vR is unimodular if v is unimodular. The matrices *u* and $\underline{v}$ found by the Hermite algorithm are unimodular, therefore the pair *u* and vR are unimodular and by direct verification we are convinced that they satisfy the representation (1.15).

Note that the set of pairs satisfying (1.15) is not exhausted by the pairs *u* and $v' = vR$. The ambiguity of the construction of *u* and *v* in the Hermite algorithm arises at step 1 due to the fact that it is possible to choose a minimal element among several equal ones. Each such pair of unimodular matrices *u* and

v corresponds to a particular solution of system (1.12), among which there is not necessarily a non-negative integer solution of the corresponding system of equations.

Applying the Hermite algorithm directly to the analysis and solution of the problem of maximizing the linear form (1.1) on the set of non-negative integer solutions of the system (1.12), we note the usefulness of the following circumstance [29].

Let the matrix of system (1.12) have rank r. Then, having solved system (1.13) in integers using formula (1.14), we obtain a new problem with n-r arbitrary integer variables $y_{r+1},...,y_n$. Such a transition, in contrast to the usual elimination of variables, which leads to rounding errors, occurs accurately, since the calculations are made with integers. In addition, such a transition can show the insolvability of the original problem in integers before the application of a special algorithm for its solution. This circumstance is even more important if we take into account that the partly applied branch and bound method works poorly in the case of the absence of an integer non-negative solution to the original problem, although the corresponding continuous problem may have a solution.

CHAPTER 2. APPROXMATE SOLUTIONS OF PROBLEMS WITH SEPARATING SETS OF CONSTRAINTS

In integer linear programming, exact algorithms were developed at the beginning of its development (for example, the cutting method and the branch and bound method). With the advent of the number of iterations by cutting methods, combined and others, approximate algorithms [6,17,25-27,28-44,105] began to appear more often, which turned out to be much more useful in practice than exact ones. The existing shortcomings of approximate algorithms (low convergence rate, lack of an optimality criterion) led to the fact that at present there is a parallel development of both exact and approximate methods.

There are many ideas for constructing algorithms for approximate solutions to integer linear programming problems. For example, the following algorithm uses a random number generator to solve the maximization problem

$$\sum_{j=1}^{n} c_j x_j$$

under restrictions

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i,\ i = \overline{1,m}$$

with boolean variables $x_j,\ j = \overline{1,\mathrm{n}}$ and non-negative coefficients $a_{ij}, b_i, c_j,$ $i = \overline{1,m};\ j = \overline{1,n}.$ The solution to the problem begins with fixing the value of the objective function b_0 and a certain number r (the number of iterations). No more than r iterations are performed to find an arbitrary admissible solution to the inequality system

$$\sum_{j=1}^{n} c_j x_j \ge b_0 \tag{2.1}$$

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i,\ i = \overline{1,m} \tag{2.2}$$

$$x_j \in \{0,1\},\ j = \overline{1,n} \tag{2.3}$$

The essence of one iteration is as follows. Let x^0 be an arbitrary set of zeros and ones. At the $(k+1)$-th iteration, the vector x^{k+1} is found by x^k. For this, the quantities are calculated

$$\Delta_0 = \max((b_0 - \sum_{j=1}^{n} c_j x_j^k) / b_0, 0),$$

$$\Delta_i = \max((\sum_{j=1}^{n} a_{ij} x_j^k - b_i) / b_i, 0),\ \mathrm{i} = \overline{1,m}.$$

The components of vector x^{k+1} are obtained from the components of vector x^k by changing each of them with the same probability p,

$$p = \min_{0 \le i \le m} \left(\rho, \max \Delta_i \right),$$

where ρ is some constant from the interval (0,1). In the given calculations, when solving practical problems, the authors took ρ=1/2. If a feasible solution to problem (2.1)-2.3) is not found after r iterations, then a conclusion is made about the non-existence of the solution and the calculation is terminated. If a feasible solution $\bar{x}$ is found, then b_0 is replaced by

$$\sum_{j=1}^{n} c_j \bar{x}_j + s,$$

where s is some positive number chosen on the basis of experience. With respect to the changed problem, r iterations are again made, etc., until a feasible solution acceptable for practice is obtained. One possible extension of this algorithm to mixed integer linear programming problems using a more targeted random vector generation method is discussed in the following sections. Later, approximate algorithms appeared, described in the works [6,26] and others. An interesting idea is to model an algorithm of the "devouring" type, in which aggregate estimates of the variables are first calculated. (The way these estimates are calculated is of great importance.) If v_j is defined, then the variables x_j are arranged in ascending order of values $c_j / v_j,\ j = \overline{1,n}.$

An approximate solution is obtained by successively assigning 1s to variables with larger estimates. A similar idea is used in another algorithm, in which an effective gradient is calculated for each variable. The variable with the largest gradient takes the value 1. The assignment of 1s occurs before the constraints of the problem are violated.

In integer programming with Boolean variables, the penalty function method is used. In addition to the terms that make up the objective function, the penalty function to be minimized includes the term

$$\sum_{j=1}^{n} k_j x_j^p (1 - x_j^p),$$

typically $p = 1$ or 2, to account for boolean variables x_j with penalty k_j. The iterative process (in Section 2.1) also uses the idea of penalty functions with exponent $p = 1$.

The methods of the decay vector and the sequential analysis and selection of variants are widely developed in numerous works by Mikhalevich and Sergienko [1,20,21,24,25,26,27]. The general scheme of the decay vector method includes procedures for finding a local minimum of the objective function $f(x)$ relative to the neighborhood $O(x^h, \rho)$ of radius ρ at point x^h, which with each step h gives a smaller value of *f(x)*, $f(x^{h+1}) < f(x^h)$.

To solve a specific discrete programming problem (linear or nonlinear), the method of sequential analysis of variants and their elimination determines the rules for selecting a set of partial solutions and their development, as well as the choice of tests for eliminating partial solutions that cannot be completed to acceptable (and even more so optimal) solutions.

The method of constructing a sequence of plans by Emelichev [3,4,5] has found significant application. The theorem presented in this work on the solution of a linear programming problem with one constraint and variables bounded from above is used further in presenting a procedure for the approximate solution of an integer linear programming problem with positive coefficients.

The packages DISPRO [21], Z – system, MATLAB, Python, Maple and numerous computing programs [65,71,110,117] have allowed to accumulate a great deal of experience in solving discrete optimization problems [63,64,96,99,109, 127,128,131]. The algorithms included in them will be further improved, providing users with more and more opportunities in solving problems that arise in practice.

In [28] the problem of partially boolean programming is considered. For it, the process of finding an approximation is described, based on minimizing a finite number of functions without restrictions by random search with local optimization on a line. Probabilistic theorems on convergence are proved. The proposed algorithms can be structurally adapted to a specific problem, taking into account its special type. This allows accelerating the calculation and

developing an effective vector generator that satisfies all or part of the constraints. This improves the search for an approximate solution. The development of algorithms and programs that take into account the specifics of problems, save the amount of stored information and accelerate the search for an optimal (suboptimal) solution is the goal of the studies described in the following sections of this chapter.

2.1. Iterative process of solving problems with boolean variables

Let it be required to find the maximum of the function

$$\sum_{j=1}^{n} c_j x_j$$

under restrictions

$$\sum_{j=1}^{n} a_{ij} x_j w_i b_i,\ i = \overline{1,m}$$

$$0 \le x_j \le d_j,\ j \in N = \{1,...,n\}$$

$$\mathrm{x}_j \in \{0,1\},\ j \in K \subset N,$$

where w_i is a relation of the type "$\le$", "$=$" or "$\ge$" and d_j is a known upper limit of the change in x_j.

The solution to this problem is found by solving a finite sequence of problems $T_0, T_1, ..., T_r$. Problem T_s, $0 \le s \le r$, contains $n_1 (\ge n)$ variables and consists of finding a feasible solution $x^s = (x_1^s, ..., x_{n_1}^s)$, the original problem and the constraint

$$\sum_{j=1}^{n} c_j x_j \ge M^s,$$

$$M^s = \begin{cases} -R_s, & s = 0 \\ \sum_{j=1}^{n} c_j x_j^{s-1} + h^s, & s > 0 \end{cases}$$

where R_s is a sufficiently large positive number, x^{s-1} is a vector found when solving problem T_{s-1}, $h^s (> 0)$ is the accuracy of the solution, which can vary depending on s.

The solution to problem T_s, $s = 0,1,2,...$ is found using an iterative process of minimizing a non-negative function $f_s(x)$. This function is such that if $\min f_s(x) = f_s(x^s) = 0$ on the set *D*,

$$D = \{x = (x_1,...,x_{n_1}) | 0 \le x_j \le 1,\ j \in K;\ 0 \le x_j \le d_j,\ j \in N \setminus K;\ x_j \ge 0,\ j > n\},$$

then x^s is its solution. In fact, we use a weakened requirement: a natural number k and a sufficiently small positive number ε are fixed. If during t iterations $(t \le k)$ such a vector x^s is found that $f_s(x^s) \le \varepsilon,$, then it is considered that the problem T_s is solved (solvable). In this case, a transition is made to solving the next problem T_{s+1} and, if T_{s+1} is unsolvable, then the solution equal to is taken as an approximate solution to the original problem

$$\bar{x}_j = \begin{cases} \left[x_j^s + 1 \setminus 2 \right], & j \in K \\ x_j^s, & j \notin K \end{cases}$$

and the process ends. If $s = 0$ and during *k* iterations this process finds a vector x^0 such that $f_0(x^0) > \varepsilon,$ then as an approximate solution of the original problem, an approximate solution of the problem T_0 is taken, calculated according to (2.8) by replacing the vector x^s with x^0.

2.1.1. Description of the iterative process for solving the problem T_s

Let us introduce the notation $m_1 = m + 1, b_0^s = M^s$,

$$r_i = \begin{cases} -1, & w_i \to "\ge" \\ 0, & w_i \to "=" \\ 1, & w_i \to "\ge" \end{cases}$$

Let us assume that the problem is being solved for the maximum, i.e. $r_0 = -1$ and $r_i = 1, i = \overline{1,m},$ are all relations w_i in (2.5) of the type "$\le$". Then $n_1 = n + m + 1 = n + m_1$.

The matrix M is introduced $p = \left\{ p_{ij} \right\},\ i = \overline{1,m_1};\ j = \overline{1,n_1}$

$$p = \begin{pmatrix} c_1 & c_2 & \dots & c_n & r_0 & \dots & \dots & \dots \\ a_{11} & a_{12} & \dots & a_{1n} & \dots & r_1 & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} & \dots & \dots & \dots & r_m \end{pmatrix}$$

and vector $b^s = \{b_i^s\}$, $i = \overline{1, \mathrm{m}_1}$,

$$b^s = (b_0^s, b_1, ..., \mathrm{b}_m).$$

Let's construct the following function

$$f_s(x) = \sum_{i=1}^{m_1} (b_i^s - \sum_{j=1}^{n_1} p_{ij} x_j)^2 + \sum_{j \in K} v_j x_j (1 - x_j),$$

where $v_j (> 0)$ is a given penalty for failure to meet the requirement of x_j being an integer, obtained using some heuristic rule or random (positive) number generator.

Lemma 2.1. Every vector x^s such that

$$\min_{x \in D} f_s(x) = f_s(x^s) = 0,$$

is the solution to the problem T^s.

Proof. If $f_s(x^s) = 0$ and $x^s \in D,$, then

$$0 \le x_j \le 1, j \in K;\ 0 \le x_j \le d_j, j \in N \setminus \mathrm{K}; \mathrm{x}_j \ge 0, j > n, \text{ и}$$

$$\sum_{i=1}^{m_1} (b_i^s - \sum_{j=1}^{n_1} p_{ij} x_j)^2 = 0,$$

$$\sum_{j \in K} v_j x_j (1 - x_j) = 0.$$

From these conditions it follows that $x_j = 0$ or 1, $j \in K$. Since the variables are non-negative, then the equalit

$$\sum_{j=1}^{n} p_{ij} x_j = b_i^s - \sum_{j=n+1}^{n_1} p_{ij} x_j = b_i - p_{ii} x_{n+i}$$

takes the form: if $i = 0$

$$\sum_{j=1}^{n} c_j x_j \ge M^s,$$

and with $i \ge 1$

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i,$$

because $p_{n+i,i} = 1$. This means that the constraints of problem T_s are satisfied and Lemma 2.1 is proven.

The solution to problem T_s, according to Lemma 2.1, consists of minimizing the function $f_s(x)$ on the set *M*. For this, a sequence of vectors $x_1^1, x_1^2, ..., x_1^k$ is constructed such that

$$f_s(x_1^p) \geq f_s(x_1^q),$$

for $p < q$,

$$x_1^1 = x_2^1,$$

$$x_1^{t+1} = \alpha^t x_1^t + (1-\alpha^t)x_2^t,\ t = \overline{1,k}, \tag{2.9}$$

where x_2^t is a random vector on the set D, $t = \overline{1,k}$, and the parameter α^t is selected so that $x_1^{t+1} \in D$ and so that the function $f_s(x)$ reaches a minimum at the point x_1^{t+1} on the segment connecting x_1^t and x_2^t.

Let ρ be a sufficiently large number. Let us introduce the notation

$$\alpha = \alpha^t, x = x_1^t, y = x_2^t, u = x - y, N_1 = \{1, ..., n_1\}$$

and calculate the numbers α_1 and α_2:

$$\alpha_1 = \max(-\rho, \rho_1^1, \rho_2^1, \rho_3^1),$$

where

$$\rho_1^1 = \max_{j \in N_1, u_j > 0}(-y_j)/u_j,$$

$$\rho_2^1 = \min_{j \in N \setminus K, u_j < 0}(d_j - y_j)/u_j,$$

$$\rho_3^1 = \max_{j \in K, u_j < 0}(1 - y_j)/u_j),$$

$$\alpha_2 = \min(-\rho, \rho_1^2, \rho_2^2, \rho_3^2), \tag{2.10}$$

$$\rho_1^2 = \min_{j \in N_1, u_j < 0}(-y_j)/u_j,$$

$$\rho_2^2 = \min_{j \in N \setminus K, u_j > 0}(d_j - y_j)/u_j,$$

$$\rho_3^2 = \max_{j \in K, u_j > 0}(1 - y_j)/u_j).$$

Lemma 2.2. For relation $x_1^{t+1} \in D$ to hold, it is necessary and sufficient that inclusion $\alpha \in [\alpha_1, \alpha_2]$ holds.

The proof follows from the rule for calculating α_1 and α_2 according to (2.10).

Let us write out formulas for calculating the following quantities:

$$l_i = \sum_{j \in N_1} p_{ij} x_j,\ s = \sum_{j \in N_1} p_{ij} y_j,\ i \in M_1$$

$$A = \sum_{i \in M_1} \left(l_i - s_i\right)^2 - \sum_{j \in K} v_j u_j^2 \tag{2.11}$$

$$B = \sum_{i \in M_1} \left(b_i^s - s_i\right)(l_i - s_i) + \sum_{j \in K} v_j (y_j - 1/2) u_j$$

$$C = \sum_{i \in M_1} \left(b_i^s - s_i\right)^2 + \sum_{j \in K} v_j y_j (1 - y_i)$$

and introduce the function

$$\varphi(\alpha) = A\alpha^2 - 2B\alpha + C. \tag{2.12}$$

Lemma 2.3. The exact value of α^t for determining x_1^{t+1} is found from the condition

$$\min_{\alpha \in [\alpha_1, \alpha_2]} f_s(\alpha x + (1-\alpha) y) = f_s(\mathrm{x}_1^{t+1})$$

and equal

$$\alpha^t = \begin{cases} \alpha_1, \left(A > 0\right) \wedge (B/A < \alpha_1)) \vee (\varphi(\alpha_1) \le \varphi(\alpha_2)) \wedge A \le 0) \\ B/\mathrm{A},\ A > 0 \wedge (\alpha_1 \le B/A \le \alpha_2) \\ \alpha_2, \left(A > 0\right) \wedge (B/A > \alpha_2)) \vee (\varphi(\alpha_1) > \varphi(\alpha_2)) \wedge A \le 0) \end{cases} \tag{2.13}$$

Proof. In the notations introduced above

$$f_s(x_i^{t+1}) = f_s(\alpha x + (1-\alpha) y) = f_s^1 + f_s^2,$$

where

$$f_s^1 = \sum_{i=1}^{m_1} \left(b_i^s - \sum_{j=1}^{n_1} p_{ij} (\alpha x_j + (1-\alpha) y_j) \right)^2$$

$$f_s^2 = \sum_{j \in K} v_j (\alpha x_j + (1-\alpha) y_j)(1 - \alpha x_j - (1-\alpha) y_j).$$

The right-hand side of the last equality contains a polynomial in α, which has the form (2.12) under the notation (2.11). Since the choice of α must be such that $x_1^{t+1} \in D$, then, based on Lemma 2.2, $\alpha \in [\alpha_1, \alpha_2]$.. It follows that the optimal value of α^t, for which

$$f_s(x_1^{t+1}) = \min_{\alpha \in [\alpha_1, \alpha_2]} \varphi(\alpha),$$

is found from condition (2.13).

Thus, the sequence $x_1^t, t=\overline{1,k}$ is defined. If $b(x)$ is a generator of random vectors x from the domain D, k is a given natural number, ε is a sufficiently small positive number, then Algorithm 2.1 for an approximate solution of problem T_s is as follows.

Algorithm 2.1.

Step 1. Formation of function $f_s(x)$.

Step 2. Generation of random vector $b(x)$ using sensor x_1^1 and assignment of $t=1$.

Step 3. Generate random vector x_2^t.

Step 4. Calculate

$$\alpha_1, \alpha_2, A, \mathrm{B}, \mathrm{C}, \alpha, \varphi(\alpha), x_1^{t+1},$$

using (2.9) - (2.13).

Step 5. If $\varphi(\alpha) \le \varepsilon$, then go to step 7.

Step 6. Increase t by one and if $t \le k$, then go to step 8.

Step 7. Form an approximate solution $\bar{x}^s$ of problem T_s:

$$\bar{x}_j^s = \begin{cases} [x_{1j}^t + 1/2], & j \in K \\ x_{1j}^t, & j \notin K \end{cases}$$

Step 8. End of the count.

If we solve the sequence of problems T_s $(s \le r)$, we will obtain a solution to the original problem. For each s, the number k of iterations is fixed. With a sufficiently large k, the problem T_s will be solved exactly or an approximate solution will be found.

Note 2.1. In the process of solving some problems, it is useful to periodically, i.e. when $t = p, 2p, ..., p\left[\frac{k}{p}\right]$ (p is a natural number greater than one), form x_2^t not randomly, but using previous calculation results.

Let the vector be $y^0 = 0$. We find y_j^q and y_j^{q+1},

$$y_j^q = \begin{cases} [x_{1j}^{pq} + 1/2], & j \in K \\ x_{1j}^{pq}, & j \notin K \end{cases}$$

$$y_j^{q+1} = \begin{cases} [x_{1j}^{p(q+1)} + 1/2], j \in K \\ x_{1j}^{p(q+1)}, j \notin K \end{cases}$$

Then, if $y_j^q = y_j^{q+1}$, then $x_{2j}^{p(q+1)+1} = y_j^q$ for $j \in K$. For the rest of the P components $x_{2j}^{p(q+1)+1}$ is a random number.

This remark is taken into account in Algorithm 2.1, in which the process of generating random vectors in steps 2 and 3 can be generalized by a procedure that ensures the highest probability of x_2^t falling into the neighborhood of a feasible solution to problem T_s.

Note 2.2. As an approximation to the sensor requirement $b(x)$, the vector x is modeled by the formulas

$$\underset{j \in N_1}{x_j} = \begin{cases} [s_0 + 1/2], j \in K \\ s_0 d_j, \ j \in N \setminus K \\ 0, \ t \le 0 \vee r_{i-1} = 0, \ j = n+i \\ t, t > 0, \ j = n+i \end{cases}, \qquad (2.14)$$

where

$$t = (b_i^s - \sum_{j \in N} p_{ij} x_j) / r_{i-1},$$

s_0 is a random number from the interval (0,1), determined, for example, using a mixed congruent sensor. It is known that such a sensor ensures a uniform distribution of the numbers obtained.

2.1.2 Convergence Theorem.

The probabilistic justification of Algorithm 2.1 is contained in the following Theorem 2.1.

Theorem 2.1. Let $b(x)$ be a procedure that ensures uniform distribution of the vector x in the domain D, $\overline{M}^s$ be the set of admissible solutions x^s of the problem T_s in the sense of $f(x^s) \le \varepsilon$, $\overline{x}_k^s$ be an approximate solution found by Algorithm 2.1 in k iterations, and $\mu_k = P\left\{\overline{x}_k^s \in \overline{M}^s\right\}$ be the probability that $\overline{x}_k^s \in \overline{M}^s$.

Then $\mu_k \to 1$ as $k \to \infty$.

Proof. Let h be a sufficiently small positive number and $\bar{x}_1^s \in \overline{M}^s$. The h -neighborhood of a point $\bar{x}^s$ is determined by the set Q:

$$Q = \left\{ x = (x_1, ..., x_{n_1}) \,|\, \left| \bar{x}_j^s - x_j \right| < 1/2,\ j \in K;\ \left| \bar{x}_j^s - x_j \right| < h,\ j \notin K \right\}.$$

The measure of set Q ($mesQ$) satisfies the condition

$$mesQ \geq (1/2)^u h^{n_1 - u},$$

where u is the number of elements of the set K.

Let $\bar{x}^s$ be the probability that $x \in Q$. We will show that $P\left\{ x_1^k \in \overline{M}^s \right\} \to 1$P when $k \to \infty$. To do this, we will imagine algorithm 2.1 as an experiment with k independent tests of whether the vector $x_2^t, t = \overline{1,\mathrm{k}}$ is in Q. Any vector $x_2^t, t = \overline{1,\mathrm{k}}$ that is in the Q with subsequent rounding of the components of $x_{2j}^t, j \in K,$ to the nearest integer coincides with $\bar{x}^s$. The probability ν of a random vector x_2^t falling into Q at a fixed t (in one trial) is equal to $mesQ / mesD$, due to the property of $b(x)$ to generate uniformly distributed vectors x_2^t from D. Therefore, ν is strictly positive.

It follows that

$$P\left\{ x_2^t \notin Q \right\} = 1 - \nu,\ 0 < 1 - \nu < 1.$$

Since the tests are independent, then

$$P\left\{ x_2^1 \notin Q \bigcap x_2^2 \notin Q \bigcap ... \bigcap x_2^k \notin Q \right\} = (1 - \nu)^k.$$

Now let us note that if $x_2^t \in Q, t = \overline{1,\mathrm{k}},$ is satisfied, then even more so is $x_1^k \in \overline{M}^s$, according to the construction of algorithm 2.1 and the choice of h.

This means that

$$P\left\{ x_1^k \in \overline{M}^s \right\} \geq P\left\{ x_2^t, t = \overline{1,\mathrm{k}} \right\} = 1 - (1 - \nu)^k.$$

Then $P\left\{ x_1^k \in \overline{M}^s \right\} \to 1$ at $k \to \infty$, and therefore $P\left\{ \bar{x}_k^s \in \overline{M}^s \right\} \to 1$ at $k \to \infty$i.e. Theorem 2.1 is proven.

This theorem can be generalized as follows.

Theorem 2.2. $\mu_k \to 1$ holds for $k \to \infty$ and when $b(x)$ is an arbitrary procedure that ensures a probability no less than $\nu \geq \nu_0 > 0$ of the vector x falling into Q.

Proof. In one iteration of Algorithm 2.1, the probability of the vector x_2^t falling into the neighborhood of Q is equal to

$$P\left\{x_2^t \in Q\right\} = \nu^t \geq \nu_0,$$

due to the property $b(x)$. Then

$$P\left\{x_2^t \notin Q\right\} = 1 - \nu^t.$$

The probability that none of $x_2^t, t = \overline{1,\mathrm{k}}$ will fall into Q is

$$\nu = (1-\nu)(1-\nu^2)...(1-\nu^k).$$

According to Algorithm 2.1, the probability of x_1^k getting into Q is not less than $1-\nu$, i.e. $P\left\{x_1^k \in Q\right\} \to 1$ for $k \to \infty$. From here, as in the conclusion of Theorem 2.1, it follows that $\mu_k \to 1$ for $k \to \infty$.

Remark 2.3. Theorems 2.1 and 2.2 are convergence theorems in the probability-theoretic sense. They are quite general and can be used to prove the probabilistic convergence of combinatorial optimization methods using a similar scheme.

Indeed, let all variables x_j in the original problem be Boolean. Then algorithm can be included in algorithm 2.1 if vector x_2^t is formed from vector x_1^t by changing the values of its components with probability $\rho, 0 < \rho \leq 1/2$. Since such a change ensures a positive probability of the changed vector coinciding with the optimal solution.

Theorem 2.3. There exists a class of algorithms $\left\{A_\rho\right\}$, where the parameter ρ is a real number from the interval (0,1) such that each algorithm A_ρ converges in the probability-theoretic sense to an optimal solution to the boolean programming problem with a sufficiently large number of iterations.

Proof. Let us describe the algorithm A_ρ for a specific ρ. Let $x = (x_1, ..., \mathrm{x}_n) -$ be a random vector, $x \in \{0,1\}$, $j = \overline{1,n}$ and $\rho \in (0,1)$, and let the vector x^k be constructed. Then x^{k+1} is found by the algorithm A_ρ as follows.

For each j, the component x_j^{k+1} of the vector x^{k+1} is obtained from the component x^{k+1} by changing it with probability ρ as follows:

$$x_j^{k+1} = \begin{cases} 1 - x_j^k, \text{ with probability } \rho, \\ x_j^k, \text{with probability } 1-\rho. \end{cases}$$

Now the proof of the convergence of A_ρ follows from Theorem 2.2 and similar arguments in Remark 2.3.

The practical value of Theorem 2.3 is that, considering a specific class of problems with Boolean variables, one can choose the most suitable number ρ for it (the A_ρ algorithm). Such a choice of algorithm can be made after an extensive computational experiment on various problems from this class. In practice, it is useful to use the A_ρ algorithm with ρ changing from a larger value to a smaller one in the process of solving a specific problem, which follows from the following reasoning.

Let in a completely integer problem with boolean variables the optimal solution $x = (x_1, ..., \mathrm{x}_n)$ be such that

$$x_j = \begin{cases} 0, \text{ with probability } p_1 \\ 1, \text{ with probability } q_1 \end{cases}$$

for $j = \overline{1,n}$.

After performing k iterations using algorithm 2.1, we obtain a vector $y = (\mathrm{y}_1, ..., \mathrm{y}_n)$ such that

$$y_j = \begin{cases} 0, \text{ with probability } p_2 \\ 1, \text{ with probability } q_2 \end{cases},$$

for $j = \overline{1,n}$.

Using the vector y and an arbitrary but fixed ρ, we calculate the vector $z = (\mathrm{z}_1, ..., \mathrm{z}_n)$ as follows:

$$z_j = \begin{cases} y_j, \text{ with probability } 1-\rho \\ 1-y_j, \text{ with probability } \rho \end{cases}$$

Let us find the probability $P\{z_j = 0\}$ that $z_j = 0$,

$$P\{z_j = 0\} = P\{z_j = y_j, y_j = 0\} + P\{z_j = 1 - y_j, y_j = 1\} = (1-\rho)\mathrm{p}_2 + \rho q_2 .$$

Similarly we find

$$P\{z_j=1\}=P\{z_j=y_j,y_j=1\}+P\{z_j=1-y_j,y_j=0\}=(1-\rho)\mathrm{q}_2+\rho p_2.$$

Now we can calculate the probability $P\{z_j=x_j\}$ that $x_j=z_j$,

$$\begin{aligned}
&P\{x_j=z_j\}=P\{x_j=0,z_j=0\}+P\{x_j=1,z_j=1\}=\\
&=p_1\left[(1-\rho)p_2+\rho q_2\right]+q_1\left[(1-\rho)\mathrm{q}_2+\rho p_2\right]=\\
&=p_1\left[p_2-\rho p_2+\rho q_2\right]+q_1\left[(\mathrm{q}_2-\rho\mathrm{q}_2+\rho p_2\right]=\\
&=p_1p_2+q_1q_2-\rho\left(p_1p_2+q_1q_2\right)+\rho\left(p_1q_2+p_2q_1\right).
\end{aligned}$$

Let us introduce the notation

$$a=p_1p_2+q_1q_2$$

Since $p_1+q_1=1$ and $p_2+q_2=1$, then

$$1=\left(p_1+q_1\right)(p_2+q_2)=a+p_1q_2+p_2q_1.$$

Hence $p_1q_2+p_2q_1=1-a$.

The desired probability is

$$P\{x_j=z_j\}=(1-\rho)a+\rho(1-a).$$

Finally

$$\max_{0\le\rho\le1}P\{x_j=z_j\}=\begin{cases}a,\ if\ \ a\ge1/2\\1-a,\ if\ \ a<1/2\end{cases}$$

From this relation follows

Theorem 2.4. In order for the probability of coincidence of z_j with x_j to be maximal, in the definition of z the value of the parameter ρ must be chosen close to one if $a<1/2$, and close to zero when $a\ge1/2$.

Indeed, since a is the probability of y_j coinciding with x_j, then at the initial iterations it is necessary to choose the value of the parameter ρ close to one, and at the final iterations ρ close to zero.

Often in practice a is unknown, then similarly to the Laplace criterion in the theory of decision-making under uncertainty, it is considered that two conditions ρ - close to one or ρ close to zero - have equal probabilities and, since both of them are practically unrealizable, then the average value between the extremes is taken, i.e. $\rho=1/2$ is assumed and the solution to the problem is carried out by the algorithm $A_{1/2}$.

2.1.3 On assessing the effectiveness of the iterative process

The analysis of algorithm 2.1 was carried out with the results of solving the same problems by other algorithms. According to the corresponding program for solving problems of Boolean and partially Boolean programming and its variants, taking into account the specifics of the problems being solved, test problems and real problems were solved: on maximizing the integral economic effect from the creation and use of digital methods by enterprises and industries in the planning period, on the implementation of a network of comprehensive schools, on the placement of agricultural crops [36], [40], etc.

In each experiment, the parameter values that led to the approximate solution found were specified. Test problems were selected in such a way as to take into account, if possible, various groups of restrictions and their impact on the performance of algorithm 2.1. Experience in solving test problems using algorithm 2.1 has shown ways to improve it when the problem dimension increases. One of the ways to improve it is to use a vector generation method that, using a random number generator, produces vectors x that satisfy part or all of the problem constraints. Such generators will be discussed below. Among the vectors generated in a given time, the one that gives the best solution for the objective function is selected.

Algorithm 2.1 provides the opportunity to use the vector generation method, which is of great help in solving large-scale applied problems, discussed in the next section. Each group of constraints has its own vector generator that satisfies these constraints. To develop such a generator and, ultimately, to apply the vector generation method, the following order of calculations is recommended:

a) constraints are analyzed; each application task is usually such that its constraints can be divided into two groups (we will denote them by A and B);

b) for constraint group A, it is possible to develop a generator that produces vectors that satisfy them;

c) in constraint group B (in some tasks, group B can be empty, then this point is skipped) possibilities for reducing the count are identified, taking into account their specificity;

d) feasible solutions are obtained and analyzed, an acceptable solution is selected.

Summarizing the experience of solving a large number of problems, we note that optimal integer solutions have been obtained for some of them. However, it has been shown that in order to increase the efficiency of solving

problems, it is necessary to specialize the sensor for obtaining pseudo-random vectors. Thus, already from the formation of vector x_2 in algorithm 2.1, the fulfillment of all or part of the restrictions is ensured. When solving practical problems, such a technique is substantially used.

Thus, the conducted computational experiment demonstrated the real possibility of using algorithm 2.1 to solve partially Boolean programming and indicated ways to improve the efficiency of the iterative process by generating vectors that satisfy all or part of the constraints of the problem under consideration.

It is precisely this specification of the scheme [28], developed in works [36, 40], that made it possible to solve a number of applied problems with separated sets of constraints.

As the analysis of many applied problems shows, the specificity of constraints can be used to speed up the procedure of checking the general admissibility of the generated vector of the problem being solved. Therefore, after presenting the general scheme of the iterative process of solving problems with Boolean variables, we will present several vector generators that satisfy constraints of various types.

2.2. Vector generators *GV1-GV8*.

Let us give examples of vector generators for a number of constraint types. Let $d(\alpha)$ be a random number generator α in the interval (0,1), $\lfloor a \rfloor$ - the integer part of the number a. If a uniform distribution of a is required, then $d(\alpha)$ can be taken as a mixed congruent random number generator in the interval (0,1).

The following generators *GV1* – *GV8* correspond to some types of constraints of a number of well-known problems (about a knapsack, about the placement and specialization of production, the multidimensional knapsack problem, etc. [13,15,26]).

GV1 Vector generator for constraints

$$\sum_{k=k_1+1}^{k_2} x_j = 1$$

$$\sum_{k=k_2+1}^{k_3} x_j = 1$$

...

$$\sum_{k=k_{m-1}+1}^{k_m} x_j = 1,\ x_j \in \{0,1\},$$

where $0 = k_1 < k_2 < ... < k_m,\ m \geq 2,$ let us denote it as *GV1*.

Algorithm *GV1*.

Step 0. Put $p = 0$.

Step 1. Increase p by one. Get a random number α using the sensor $d(\alpha)$.

Step 2. Find $i_p = [\alpha(k_{p+1} - k_p - 1] + k_p + 1$.

Step 3. Put $x_{i_p} = 1$, the rest $x_j = 0,\ k_p + 1 \leq j \leq k_{p+1}$.

Step 4. If $p + 1 = n,$ then end the count, otherwise go to step 1.

GV2 Constraint Vector Generator

$$\sum_{j=1}^{n} x_j \leq 1,\ x_j \in \{0,1\},\ \mathrm{j} = \overline{1,n}. \qquad (2.15)$$

Algorithm *GV2*.

Step 1. Generate number α, calculate $j_0 = \lfloor n\alpha \rfloor + 1$.

Step 2. Generate number α again and calculate

$$x_{j_0} = \begin{cases} 0, \textit{ if } 0 \leq \alpha \leq 1/2 \\ 1, \text{ otherwise} \end{cases}$$

Set the remaining $x_{j_0}\ (j \neq j_0)$ equal to zero.

Remark 2.4. In the case of equality in (2.15), in the *GV2* algorithm, step 2 would consist of calculating the coordinates of x_j using the formula

$$x_j = \begin{cases} 0, \textit{ if } j = j_0 \\ 1, \text{ otherwise} \end{cases}$$

GV3 Vector generator for constraints

$$\sum_{j=1}^{n} x_{ij} = 1$$

$$\sum_{j=1}^{m} x_{ij} = 1$$

$$x_{ij} \geq 0, \text{ integers, } i = \overline{1,m}, \ j = \overline{1,n}.$$

Algorithm *GV3*.

Step 0. Put $p = 0$, sets M and N - empty.

Step 1. Increase p by one.

Step 2. Get a random pair (i_0, j_0) such that

$$i_0 \notin M, \ j_0 \notin N, \ i_0 \in \{1,...,m\}, \ j_0 \in \{1,...,n\}.$$

Step 3. Get a random number α and calculate

$$x_{i_0 j_0} = \begin{cases} 1, \textit{ if } 1/2 \leq \alpha < 1 \\ 0, \textit{ otherwise} \end{cases},$$

i_0 to set M, and add j_0 to set N.

Step 4. If $p = n$, then end of count, otherwise go to step 1.

GV4 Vector generator for constraints

$$\sum_{j=1}^{n} a_j x_j \leq b$$

$$x_j \geq 0, \ \textit{integers}, \ j = \overline{1,n},$$

moreover $b > 0, \ a_j > 0, \ j = \overline{1,n}.$

Algorithm *GV4*.

Step 0. Put $p = 0, c = b.$

Step 1. Increase p by one. Get α with random number generator.

Calculate $d_p = \left| c / a_p \right|, \ x_p = [\alpha d_p + 1/2].$

Step 2. Replace c with $c - a_p x_p$.

Step 3. If $p < n \wedge c > 0$, then return to step 1.

GV5 Constraint-Satisfying Vector Generator

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \ i = \overline{1,\mathrm{m}}$$

$$x_j \geq 0, \ \textit{integers}, \ j = \overline{1,n},$$

and $b_i > 0, \ a_{ij} > 0, \ i = \overline{1,\mathrm{m}}; \ j = \overline{1,n}.$

Algorithm *GV5*.

Step 0. Put $p = 0, \ c = (c_1,...,c_m) = (b_1,...,\mathrm{b}_m).$

Step 1 Increase p by one, get the number α and calculate

$$d_p = \min_{1 \le i \le m}[c_i / a_{ip}],$$

$$x_p = [\alpha d_p + 1/2].$$

Step 2. Replace c_i with $c_i - a_{ip}x_p$, $i = \overline{1,m}$.

Step 3. If $p < n \wedge \min_{1 \le i \le m} c_i > 0$, then go to step 1.

GV6 Vector generator for equality constraints

$$x_1 - y_2 = d_1$$

$$x_j + y_j - y_{j+1} = d_j, \ j = \overline{2,n-1},$$

$$x_n + y_n = d_n,$$

where $x_1, x_j, y_j - integers, \ all \ge 0, \ j = \overline{2,n}$.

Algorithm *GV6*.

Step 1. A random number α is generated using sensor $d(\alpha)$ and $y_n = [\alpha d_n]$ is calculated.

We find $x_n = d_n - y_n, p = n-1$.

Step 2. A random number α is generated and determined

$$y_p = [\alpha(d_p + y_{p+1}],$$

$$x_p = d_p + y_{p+1} - y_p.$$

Step 3. Check $p = 2$? . If not, then decrease p by one and return to step 2.

Step 4. Calculate $x_1 = d_1 + y_2$.

GV7 Vector generator for constraints

$$\sum_{k=1}^{l}\sum_{j=1}^{m} u_{kj}\delta_{kj} \le c$$

$$\sum_{k=1}^{l} \delta_{kj} \le 1, \ j = \overline{1,m}$$

$$\delta_{kj} \in \{0,1\}, \ k = \overline{1,l}; \ j = \overline{1,m},$$

$c, u_{kj} \left(k = \overline{1,l}; \ j = \overline{1,m}\right)$ are non-negative numbers.

Algorithm *GV7*.

Step 1 Generate a random permutation $\{j_1, j_2, ..., j_n\}$ of numbers $\{1, 2, ..., m\}$, $p = 1, \mu = c.$ is assumed.

Step 2. Form a set $K_{jp} = \{k \mid u_{kj_p} \le \mu\}$. Select a random number k_0 from the set K_{j_p} and a random number α less than 1/2.

Calculate

$$\delta_{kj_p} = \begin{cases} 1, \text{ if } K_{j_p} \text{ non-empty, } k = k_0, \alpha < 1/2 \\ 0, \text{ otherwise} \end{cases}$$

Step 3. Replace μ with $\mu - \sum_{k=1}^{l} u_{kj_p} \delta_{kj_p}$.

If $\mu = 0,$ then the remaining undefined δ_{kj} are set to zero. End of counting.

Step 4. Increase p by one. If $p \le m$ and $\mu > 0$, then go to step 2.

GV8 Vector generator for constraints

$$\sum_{i=1}^{m} x_{ij} = 1, \; j = \overline{1,n}$$

$$\sum_{j=1}^{n} A_{ij} x_{ij} \le B_i, \; i = \overline{1,\mathrm{m}}$$

$$x_{ij} \in \{0,1\}, \; i = \overline{1,m}; \; j = \overline{1,n}.$$

No set $\{x_{kj}\}$ of zeros and ones satisfies these constraints unless there exists a $j = j_0$ such that $A_{ij_0} \le B$ holds for all $i = \overline{1,m}$. We assume that the restrictions are not contradictory.

Algorithm *GV8*.

Step 1. $p = 0, c_i = B_i, \; i = \overline{1,m}$ A is assumed and a random permutation $\{j_1, j_2, ..., j_n\}$ of numbers $\{j_1, j_2, ..., j_n\}$ is generated.

Step 2. p is increased by one. Calculate

$$d = \begin{cases} 1, \text{ if } A_{ij_p} \le c_i, \; i = \overline{1,m} \\ 0, \text{ otherwise} \end{cases}$$

Step 3. If $d = 1,$ then select a random number $d = 1$ from the set $\{1, 2, ..., m\}$.

Set $x_{i_o j_0} = 1,\ x_{ij_p} = 0,\ i = \overline{1,m};\ i \neq i_0$. If $d = 0$, then $x_{ij_p} = 0,\ i = \overline{1,m}$.

Step 4. Replace c_i with $c_i - A_{ij_p} x_{ij_p},\ i = \overline{1,m}$.

Step 5. If $p < n$ and $c_i \geq 0,\ i = \overline{1,m},$, then go to step 2.

The presented generators containing random elements are, on the one hand, tied to specific problems, on the other hand, they are quite general (for example, *GV5*). This generality is determined by the possibility of reducing a problem with a limited domain of integer solutions to a finite series of problems with positive coefficients, which is shown in the following section 2.3.

2.3 Transformations of the integer linear programming problem

The importance of the form of representation for the successful solution of a specific discrete optimization problem is noted in [13]. Two transformations of the integer linear programming problem are presented here. The first transformation is associated with the reduction of the general integer linear programming problem to problems with positive coefficients (Section 2.3.1), and the second is associated with the reduction of several linear constraints to one equivalent linear constraint. Unlike the known methods of reduction (aggregation), here we present an aggregation scheme where one equation is such that it can take integer values, and the other is such that it can take any real values. The latter result is contained in section 2.3.2. Both transformations lead to problems for each of which it is relatively easy to generate integer non-negative vectors that satisfy their constraints.

2.3.1 Reduction of a general integer linear programming problem to a series of problems with positive coefficients

Let in the maximization problem

$$\sum_{j=1}^{n} c_j x_j \tag{2.16}$$

under restrictions

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, i = \overline{1,\mathrm{m}}, \tag{2.17}$$

$$x_j \geq 0,\ integers,\ j = \overline{1,n} \tag{2.18}$$

In addition, we will assume that the region (2.17)-(2.19) is bounded, i.e. there exists an integer such that

$$0 \le \sum_{j=1}^{n} x_j \le B$$

and the coefficients $a_{ij}, b_i, c_j \left(i = \overline{1, \mathrm{m}};\ j = \overline{1, n} \right)$ are integers.

Problem (2.16) - (2.18) is reduced to a finite number of problems with positive coefficients of the form $P(b), 0 \le b \le B$, b is an integer,

$$\max \sum_{j=1}^{n} \bar{c}_j x_j$$

$$x_j \le b_i + bt_i,\ i = \overline{1, m}$$

$$\sum_{j=1}^{n} x_j \le b$$

$$x_j \ge 0,\ \textit{integers},\ j = \overline{1, n},$$

$$\bar{c}_j = c_j + M,\ \ j = \overline{1, \mathrm{n}},$$

where A is a sufficiently large number,

$$J_i = \left\{ j \mid a_{ij} \le 0 \right\},\ i = \overline{1, m}$$

$$t_i = \begin{cases} \varepsilon + \max\limits_{j \in J_i}(-a_{ij}), J_i - \textit{not empty}, \\ 0,\ J_i - \textit{empty}. \end{cases}$$

For $\varepsilon > 0 (\varepsilon = 0)$, problem $P(b)$ will have positive (non-negative) coefficients.

Theorem 2.4. Let $\bar{x}$ be an optimal solution to problem (2.16)-(2.18). The optimal solution $\bar{z}$ of problem $P\left\{ \sum_{j=1}^{n} x_j \right\}$ is an optimal solution to problem (2.16)-(2.18).

Proof. Since $\bar{x}$ is an optimal solution to problem (2.16)-(2.18), then

$$\sum_{j=1}^{n} a_{ij} \bar{x}_j \le b_i, i = \overline{1, \mathrm{m}},$$

$$\bar{x}_j \ge 0,\ j = \overline{1, n}$$

From here

$$\sum_{j=1}^{n}\left(a_{ij}+t_i\right)\bar{x}_j \le b_i+\bar{b}t_i,\ i=\overline{1,m}, \tag{2.20}$$

where $\bar{b}=\sum_{j=1}^{n}\bar{x}_j$.

Therefore, $\bar{x}$ is a feasible solution to problem $P(\bar{b})$. We denote by $\bar{z}$ the optimal solution to problem $P(\bar{b})$, x the optimal solution to problem (2.16)-(2.18). Then

$$\begin{gathered}\bar{c}\bar{x}\le\bar{c}\bar{z}\\ \sum_{j=1}^{n}\bar{z}_j\le\bar{b}\end{gathered} \tag{2.21}$$

Since M is a sufficiently large number, the equality follows

$$\sum_{j=1}^{n}\bar{z}_j=\bar{b}. \tag{2.22}$$

Indeed, if

$$\sum_{j=1}^{n}\bar{z}_j<\bar{b},$$

tnen

$$M>\sum_{j=1}^{n}c_jx_j-\sum_{j=1}^{n}c_j\bar{x}_j\ge\sum_{j=1}^{n}c_j\bar{z}_j-\sum_{j=1}^{n}c_j\bar{x}_j\ge$$

$$\ge\frac{\sum_{j=1}^{n}c_j\bar{z}_j-\sum_{j=1}^{n}c_j\bar{x}_j}{\bar{b}-\sum_{j=1}^{n}\bar{z}_j},$$

because $\bar{b}-\sum_{j=1}^{n}\bar{z}_j\ge 1.$

From here

$$M\left(\bar{b}-\sum_{j=1}^{n}\bar{z}_j\right)>\sum_{j=1}^{n}c_j\bar{z}_j-\sum_{j=1}^{n}c_j\bar{x}_j$$

$$\sum_{j=1}^{n}c_jx_j+M\bar{b}>\sum_{j=1}^{n}c_j\bar{z}_j+M\sum_{j=1}^{n}\bar{z}_j,$$

or in view of (2.20)

$$\sum_{j=1}^{n} c_j \bar{x}_j + M\sum_{j=1}^{n} \bar{x}_j > \sum_{j=1}^{n} c_j \bar{z}_j + M\sum_{j=1}^{n} \bar{z}_j,$$

i.e. $\vec{c}\bar{x} > \vec{c}\bar{z}$, which contradicts the optimality of $\bar{z}$ in the problem $P(b)$. Thus,

$$\sum_{j=1}^{n} \bar{x}_j = \sum_{j=1}^{n} \bar{z}_j \tag{2.23}$$

Subtracting equality (2.22) multiplied by t_i, from the inequality

$$\sum_{j=1}^{n} \left(a_{ij} + t_i\right)\bar{z}_j \le b_i + \bar{b}t_i,\ i = \overline{1,m}$$

we obtain that $\bar{z}$ is a feasible solution to problem (2.16) - (2.18).

We have $c\bar{x} \ge c\bar{z}$ and taking into account (2.23) we obtain

$$\vec{c}\bar{x} \ge \vec{c}\bar{z} \tag{2.24}$$

From (2.21) and (2.24) we deduce that $\vec{c}\bar{x} = \vec{c}\bar{z}$ and again from (2.23) $c\bar{x} = c\bar{z}$, i.e. $\bar{z}$ is one of the optimal solutions of problem (2.16)-(2.19). Theorem 2.4 is proven.

Let us show how the solution of problem (2.16) - (2.18) can be obtained from the solution of problems $P(b)$, $b = \overline{0,B}$. First of all, we note that if $b = 0$ or $b_i + bt_i = 0$, then problem $P(b)$ has only a trivial solution, and if for some i, $b_i + bt_i = 0$, then the problem $P(b)$ is inconsistent. Therefore, we solve only those problems $P(b)$ for which $b > 0$ and $b_i + bt_i > 0$. The number of such problems is B_1, $B_1 \le B$, and each with positive integer coefficients.

Let $\bar{z} = \bar{z}(b)$ be the optimal solution to problem $P(b)$. We check whether it is admissible for problem (2.16) - (2.18). If so, we remember the one that gives the greatest value of the objective function (2.16) - it will be the optimal solution to the original problem.

Thus the following theorem is proved.

Theorem 2.5. The solution of problem (2.16) - (2.18) is equivalent to B_1 of problems of type $P(b)$ with positive coefficients.

The transformation of problem (2.16) - (2.18) to problems with positive coefficients is of interest for the following reasons: 1) many integer linear programming algorithms require the presence of a feasible integer non-negative solution at their initial stage of operation. Here, the problem always has a

feasible solution, for example, a trivial one; 2) feasible vectors can be easily generated for the problem (using heuristic algorithms) (one such generator is described in [38], other vector generators for such a problem are described in section 2.4); 3) the transformation of problem (2.16)-(2.19) to $P(b)$ problems is also valid in the case of Boolean variables.

2.3.2 On aggregation of equations

Consider the knapsack problem

$$\max \sum_{j=1}^{n} c_j x_j \tag{2.25}$$

$$\sum_{j=1}^{n} a_j x_j \le b \tag{2.26}$$

$$x_j \ge 0,\ j = \overline{1,n} \tag{2.27}$$

$$x_j,\ integer,\ j = \overline{1,n}, \tag{2.28}$$

where c_j, a_j, b $(j = \overline{1,n})$ are integers. The problem of integer linear programming is reduced to problem (2.25) - (2.28), in which condition (2.26) is replaced by condition B.

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, i = \overline{1,m} \tag{2.29}$$

Some results on aggregation of equations are contained in the works [13]. It is known that if aggregation of equations is performed in integers, then large absolute values may arise at intermediate iterations. When changing the formulation of the question of aggregation so that it is still possible, the numbers in the aggregating equation may not increase much.

Let us formulate Lemma 2.4, and then prove that aggregation in the case of non-integer coefficients in one of the two aggregated equations is possible (Algorithm 2.2) and estimate the coefficients of the resulting equation [32].

Lemma 2.4. Let P be a set of arbitrary structure, and let functions $f_1(x)$ and $f_2(\mathrm{x})$ defined on P be such that there exists a real number λ $(\lambda < \infty)$ such that

$$\sup_{\substack{x \in P \\ f_2(\mathrm{x}) \neq 0}} \left| f_1(x) / \mathrm{f}_2(\mathrm{x}) \right| < \lambda$$

Then the system of two equations

$$f_1(x) = 0 \tag{2.30}$$

$$f_2(\mathrm{x}) = 0,\ x \in P$$

is equivalent to equation

$$f_1(x) + \lambda f_2(\mathrm{x}) = 0, x \in P \tag{2.31}$$

Proof. If x satisfies (2.30), then it also satisfies (2.31). Conversely, let x satisfy (2.31). Then if $f_2(\mathrm{x}) = 0,$ then so does $f_1(x) = 0.$ If we assume that $f_2(\mathrm{x}) \neq 0,$ then

$$\lambda = -f_1(x) / \mathrm{f}_2(\mathrm{x}) \leq \sup_{\substack{x \in P \\ f_2(\mathrm{x}) \neq 0}} \left| f_1(x) / \mathrm{f}_2(\mathrm{x}) \right|$$

We get a contradiction.

Based on Lemma 2.4, we describe the following algorithm 2.3 of aggregation (2.29). Let $m \geq 2$ and P be a set of bounded vectors,

$$x = (x_1, ..., x_n),\ 0 \leq \mathrm{x}_j \leq d_j,\ j = \overline{1, n};$$

and a_{ij}, b_i are integers.

Algorithm 2.2.

Step 0. Set i equal to zero.

Step 1. Increase i by one

Step 2. If $i = m,$ then go to step 7.

Step 3. Find

$$a_{ik} = \max_{1 \leq j \leq n} \left| a_{ij} \right|$$

Step 4. Divide both sides of the i-th equation by $\left| a_{ik} \right|$ to obtain the equation

$$\sum_{j=1}^{n} \alpha_{ij} x_j \leq \beta_i, i = \overline{1, m},$$

where

$$\alpha_{ij} = a_{ij} / \left| a_{ik} \right|,\ \beta_i = b_i / \left| a_{ik} \right|.$$

Step 5. By two equations

$$\sum_{j=1}^{n} \alpha_{ij} x_j = \beta_i$$

$$\sum_{j=1}^{n} a_{i+1,j} x_j = b_{i+1},$$

using Lemma 2.4, we construct one equation

$$\sum_{j=1}^{n} \alpha'_{i+1,j} x_j = \beta'_{i+1},$$

where

$$\alpha'_{i+1,j} = \alpha_{ij} + \lambda a_{i+1,j}$$

$$\beta'_{i+1} = \beta_i + \lambda b_{i+1}$$

$$\lambda > \max_{0 \le x_j \le d_j} \left| \sum_{j=1}^{n} \alpha_{ij} x_j \right| + |\beta_i|$$

Step 6. We make a replacement

$$a_{i+1,j} = \alpha'_{i+1,j}, b_{i+1} = \beta'_{i+1}$$

and return to step 1.

Step 7. End of calculation; the aggregate equation is represented by the coefficients

$a_{m1}, ..., a_{mn}$ and b_m.

Lemma 2.5. Each coefficient of the equations obtained during the operation of algorithm 2.2 does not exceed the value $2rs + r + s$,

where

$$\mathrm{r} = \sum_{j=1}^{n} d_j,$$

$$s = \max(s_1, s_2),$$

$$s_1 = \max_{\substack{1 \le i \le m \\ 1 \le j \le n}} |a_{ij}|, \; s_2 = \max_{1 \le i \le m} |b_i|.$$

Proof. Any $|\alpha_{ij}| \le 1, |a_{ij}| \le s$ and $|b_i| \le s$,

$$|\beta_i| \le \max_{0 \le x_j \le d_j} \left| \sum_{j=1}^{n} \alpha_{ij} x_j \right| \le \sum_{j=1}^{n} |\alpha_{ij}| \max_{1 \le i \le m} |b_i| \le \sum_{j=1}^{n} d_j.$$

Next, we will estimate the numbers $\alpha'_{i+1,j}, \beta'_{i+1}$ and λ, considering that

$$\lambda = \max_{0 \le x_j \le d_j} \left| \sum_{j=1}^{n} \alpha_{ij} x_j \right| + |\beta_i| + 1.$$

We receive

$$\lambda < 2\sum_{j=1}^{n} d_j + 1 = 2\mathrm{r}+1,$$

$$\alpha'_{i+1,j} \le |\alpha_{ij}| + \lambda |a_{i+1,\mathrm{j}}| \le 1 + (2r+1)s = 2rs + s + 1,$$

$$\beta'_{i+1} \le |\beta_i| + \lambda |b_{i+1}| \le r + (2r+1)s = 2rs + s + r.$$

It is natural to assume that $r \ge 1,$ since otherwise all x_j would be equal to zero. Lemma 2.5 is proved.

Thus, the moduli of the coefficients of the equation aggregated according to Algorithm 2.2 have an upper bound equal to $2rs + r + s.$. This result does not contradict the fact that when aggregating equations with integer coefficients, the intermediate numbers grow in absolute value, since Lemma 2.4 allows one to aggregate two linear equations, in one of which the coefficients may be non-integer.

2.4 An approximate algorithm for solving an integer programming problem with positive coefficients.

We describe the procedure P for an approximate solution of $P(b)$ for fixed $b,\ b > 0,$ and $b_i + bt_i > 0,\ i = \overline{1,m}$.

Procedure P.

To solve the problem

$$\max \sum_{j=1}^{n} c_j x_j$$

under restrictions

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, i = \overline{1,m}$$

$$x_j \ge 0,\ integer,\quad j = \overline{1,n},$$

where $\mathrm{a}_{ij}, b_i, c_j \left(i = \overline{1,m};\ j = \overline{1,n} \right)$ are positive integers.

Step 1. Put

$$F = 0,\ x^* = (0,...,0).$$

$$F_{\max} = 0, x_{\max} = (0,...,0).$$

Step 2. Find the upper integer bounds $\bar{x}_j$ for the variables x_j,

$$\bar{x}_j = \min_{1 \le i \le m} [b_i / a_{ij}], \; j = \overline{1,n}$$

Step 3. If $\bar{x}_j = 0, \; j = \overline{1,n},$ then go to step 11.

Step 4. Generate (using the methods described below) a feasible vector z with the value s of the objective function, $s = cz.$

Step 5. Find $\bar{z}_k, k = \overline{1,n}$ from the solution to the problem

$$\bar{z}_k = \max \sum_{j=1}^{n} c_j x_j$$

under restrictions

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, i = \overline{1,m}$$

$$0 \le x_j \le \bar{x}_j, \; j \ne k$$

$$x_k = 0$$

Step 6. Put

$$\theta = 1 + \max\left(\min_{1 \le j \le n} R_j, s\right),$$

where $R_j = \left[\bar{z}_j\right]$ and $y_j = 0$ at $\theta \le \bar{z}_j.$

Step 7. For each j such that $\theta > \bar{z}_j$ and $\bar{x}_j > 0$ find

$$y_j = \min x_j$$

under restrictions

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, i = \overline{1,m}$$

$$\sum_{j=1}^{n} c_j x_j \ge \theta$$

$$0 \le x_j \le \bar{x}_j, \; j = \overline{1,n}.$$

If the constraints of one of these problems (and therefore all of them) are contradictory or the found integer non-negative vector

$$\underline{x} = (\underline{x}_1, ..., x_n) = \left(\right]y_1[, ..., y_1[\right)$$

does not meet the constraints

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, i = \overline{1,m},$$

then in case $F_{\max} < F + s$, calculations are performed

$$F_{\max} = F + s, x_{\max} = x^* + z,$$

and go to step 11.

Step 8. Find

$$F := F + \sum_{j=1}^{n} c_j \underline{x}_j$$

$$x_j^* := x_j^* + \underline{x}_j, j = \overline{1,n}$$

If $F > F_{\max}$, then $F_{\max} = F, x_{\max} = x^*$.

Step 9. Recalculate the right-hand sides of the constraints

$$b_i := b_i - \sum_{j=1}^{n} a_{ij} \underline{x}_j, \; i = \overline{1,m}.$$

Step 10. If $b_i > 0, i = \overline{1,m},$ then go to step 2.

Step 11. End of the count; $x_{\max}$ is an approximate solution to the problem with the value $F_{\max}$ of the objective function.

The procedure P is presented in full, but step 4 requires an explanation of the method for generating a feasible solution. Since a problem with positive coefficients always has a feasible, for example, trivial solution, then the generation of a feasible solution x should be done, firstly, with the largest possible value cx of the objective function and, secondly, quite simply.

For these purposes, the following deterministic generators of admissible solutions are applicable: a) the optimal continuous solution $x = (x_1, ..., x_n)$, corresponding to the problem $P(b)$ is found. Then the solution $y = ([x_1], ..., [x_n])$ obtained by taking the integer part of the coordinates of the vector x is an admissible solution to the problem $P(b)$, due to the positivity of its coefficients; b) obtaining an admissible solution in the following way. Initially calculating $N = \{1, ..., n\}$, repeat the following steps 1-2 no more than n times:

Step 1. Find the number $k \in N$ at which

$$\frac{c_k b_l}{a_{lk}} = \max_{1 \le j \le n} \frac{c_j}{h_j},$$

where $h_j = \max_{1 \le i \le m} \frac{a_{ij}}{b_i}$, define

$$g_k = \max_{1 \le i \le m} [b_i / a_{ik}],$$

replace b_i with $b_i - a_{ik} g_k$ and N with $N - \{k\}$.

Step 2. If all $b_i > 0$, $i = \overline{1,m}$ and $N \not\in 0$, then go to step 1, otherwise end of count.

The positivity of the coefficients $P(b)$ allows us to construct a whole series of other deterministic generators of admissible solutions, as well as generators that use elements of randomness, as, for example, in [38,41].

A number of operations of procedure P in the case of the knapsack problem are naturally simplified. In this case, a significant simplification occurs in the process of generating a feasible integer solution at step 4 and in solving auxiliary linear programming problems at steps 5 and 7.

Here we use the theorem [18] that the solution $\{g_j\}$ of the problem

$$\max \left\{ \sum_{j=1}^{n} c_j x_j \mid \sum_{j=1}^{n} a_j x_j \le b, 0 \le x_j \le d_j, j = \overline{1,n} \right\},$$

the variables of which are ordered in non-increasing order of the ratios of the c_j / a_j, $i = \overline{1,n}$ are calculated using the formula

$$g_j = \min\left(d_j, t / a_j\right), \ j = \overline{1,n},$$

where

$$t = \begin{cases} b, \text{ if } j = 1 \\ b - \sum_{k=1}^{j-1} a_k g_k, \text{ if } j > 1 \end{cases}$$

For the knapsack problem with boolean variables, $d_j = 1$, at step 7, positive values of y_j correspond to positive values of x_j, i.e. $x_j = 1$. By fixing these variables, we can reduce the dimensionality of the original problem. A

sign of the positivity of y_l (with the solvability of the l-th problem at step 7) is that the value of μ, equal to

$$\max \sum_{j=1}^{n} c_j x_j$$

under restrictions

$$\sum_{j=1}^{n} a_j x_j \le b, 0 \le x_j \le \bar{x}, j = \overline{1,n}, x_l = 0,$$

will be less than θ.

Let us present some theorems concerning the properties of procedure P.

Theorem 2.6. If each execution of step 4 is carried out with the value $\theta = 1 + s$, then procedure P produces an exact solution to the integer linear programming problem with positive coefficients.

Proof. Let each execution of step 4 be carried out with $\theta = 1 + s$. This means that $s \ge \min_{1 \le j \le n} R_j$, i.e. there is a number q such that $s \ge R_q$. Consequently, the objective function on the found feasible solution has no less value than on the optimal solution of the maximization problem

$$\sum_{j=1}^{n} c_j x_j$$

under restrictions

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, i = \overline{1,m}$$

$$0 \le x_j \le \bar{x}, j = \overline{1,n}$$

$$x_q = 0, q \in \{1, ..., n\}$$

Then, solving the problem of finding the lower boundaries of the $\underline{x_j}$ variables x_j, $j = \overline{1,n}$, at step 4 of the procedure P with $i = q$ we will have $\min x_q > 0$ and $]y[> 1$. Therefore, $\underline{x}$ will be non-zero and a shift in the origin of coordinates will necessarily occur (step 9 is performed), i.e. after no more than B shifts of the origin of coordinates, the process will end with obtaining an optimal solution to problem (2.16)-(2.19) with positive variables.

Theorem 2.6 contains one of the sufficient conditions for obtaining an exact solution by the procedure P. This condition is always satisfied for $n = 2$.

Theorem 2.7. For $n = 2$, the procedure P generates an exact solution to problem (2.16) - (2.19) with positive coefficients.

Proof. The calculation according to step 4 of procedure P leads at $n=2$ to the construction of a non-zero vector for any right-hand side of the constraints (if, of course, a non-zero admissible vector exists). For this, it is sufficient to use one generator b). The use of other generators together with it can lead to an improvement in the admissible solution.

Therefore, the points $\left(0,\bar{x}_2\right)$ and $\left(0,\bar{x}_2\right)$ lie below the line $cx=\theta$ corresponding to the inequality $c_1x_1+c_2x_2\geq 0,$ i.e. $c_2\bar{x}_2<\theta,\ c_1\bar{x}_1<\theta..$ This means that at $n=2$ at step 4, the value of θ is equal to $\theta=1+s.$

Then, according to Theorem 2.6, the procedure produces an exact solution to the problem specified in Theorem 2.7.

Theorem 2.8. The number of operations and the amount of memory required to implement procedure P have orders $\mathrm{O}\left(\left(mn+l\right)n^2M\right)$ and $O\left(mn^2\right)$, respectively, where l is the complexity of solving a linear programming problem of order $(m+1)n$ with variables bounded from above, $M=\max\left\{a_{ij},b_i,c_j\right\},\ i=\overline{1,m};\ j=\overline{1,n}.$

The proof is carried out by direct calculation according to the description of the procedure P. It is known that the value l is polynomial in the length of the input information (and therefore in time) [54], therefore the procedure P is pseudo-polynomial in time.

Computational experiments conducted with the procedure P, when solving integer linear programming problems with positive coefficients, yielded the following results. 20 problems were solved with $m=6-10,\ n=6-10,a_{ij}=10-25,$ For each of the three intervals of change of the right-hand sides: $b_i=100-150,\ 200-250,300-350.$

The solutions obtained by the procedure P, in most cases, coincided with the known optimal solutions of these problems.

CHAPTER 3. ALGORITHMS AND PROGRAMS FOR DISCRETE AND COMBINATORIAL OPTIMIZASTYION

The section discusses various aspects and approaches regarding the search for a system of differenrt representatives (*SDR*) of subsets of a finite set. Algorithm 3 has been developed, which is an algorithmization of the search process for *SDR* on the basis of Hall's theorem on the existence of *SDR*. Algorithms 1-5 for searching for partial and complete *SDRs* based on the technology for solving problems of the transport type, in particular, the algorithm associated with the construction of cycles, are given. *SDR* search algorithms can be used as auxiliary tools for solving combinatorial problems. It is also shown, conversely, that the search for *SDR* can be considered as a combinatorial optimization problem [140,143].

3.1 Search Algorithms for a System of Different Representatives of Subsets of a Finite Set

Let S be a non-empty finite set, $M = (S_1, S_2, ..., S_m)$ −some sequence of its non-empty subsets. A system of different representatives (*SDR)* for a collection of sets *S* is a set $a = \{a_1, a_2, ..., a_k\}$ such that ai belongs to S_i and for any *i* and *j* $a_i \neq a_j$. An *SDR* for any subsequence of the sequence *M* is said to be partial.

The need to construct a system of distinct representatives (*SDR*) arises both in purely combinatorial mathematical studies and in their applications to linear programming, mathematical economics, and combinatorial optimization. Within the framework of combinatorial mathematics, *SRPs* are mainly used in solving problems of choice and optimization, in particular, in studying Latin rectangles [92], in the assignment problem, and in studying special matrices with non-negative elements. At the same time, the analysis of studies on *SDR* shows that the task of developing *SDR* search algorithms, especially algorithms that build *SDR* with certain properties, is relevant. In this regard, the algorithms 1-5 presented in the paper can be useful. Let us give examples of practical interpretation of *SDR*:

a) There are *m* commissions in some institution. It is required from the composition of each commission to appoint their chairmen so that no one person presides over more than one commission. Here the *SDRs* of the commissions will be composed by their chairmen;

b) There are *n* vacancies in the company, each of which requires a certain qualification. There are m people who apply for *n* jobs. What is the largest number of vacancies that can be filled from the pool of applicants, if the vacancy can only be filled by a person corresponding to his qualifications. In this problem, the interesting question is the largest number of jobs that can be filled by qualified candidates and the specific assignment of the largest number of applicants to the jobs they apply for;

c) Suppose we are given a set of hours per week, a set of *n* teachers, each of whom has the opportunity to teach during some subset of hours, a set of *m* classes, each of which has only a few hours per week in the schedule, and an *nxm* matrix of non-negative integers numbers, in which the (*i,j*)-th element means the number of hours during which the *i*-th teacher must teach in the *j*-th class. It is required to determine whether it is possible to schedule lessons and distribute teachers in such a way that each class met at the required hours, each teacher could be in his classes, there would be only one teacher per class and no teacher would be assigned to two classes at the same time.

In problems of type a)-b), it is required to determine a complete system of different representatives or a partial system, but with the maximum number of representatives. When studying the problems associated with *SDR*, the practical issue of algorithmization of the process of searching for such systems is relevant.

The system of different representatives consists of elements a_i, one from each subset S_j In case

$$S = \{S_1, S_2, S_3, S_4\} = \{\{s_1, s_4, s_5\}, \{s_1\}, \{s_2, s_3, s_4\}, \{s_2, s_4\}\}$$

one can define four *SDR*:

$$a_1 = \{s_1, s_2, s_3, s_4\}, \quad a_2 = \{s_1, s_2, s_3, s_5\},$$
$$a_3 = \{s_1, s_3, s_4, s_5\}, \quad a_4 = \{s_1, s_2, s_4, s_5\}.$$

SDR exists only if for any *k* subsets S_j their union contains at least *k* elements[140]. For example, in the family of subsets S_1, S_2, S_3, S_4, S_5, sets

$$S=\{\{s_1, s_2\}, \{s_1, s_3\}, \{s_2, s_3\}, \{s_1, s_2, s_3\}, \{s_1, s_3, s_4, s_5\}\},$$

unable to find five distinct s_i elements one from each S_j, so this family has no *SDR*. Here the union of four sets $\bigcup_{i=1}^{4} S_i$ contains only three elements $\{s_1, s_2, s_3\}$.

The idea of replacing sets with their representatives turned out to be fruitful and was further developed. Systems of representatives are singled out taking into account the conditions of the tasks or the goals of theoretical generalizations. For example, task research about splitting sets, led to the concept of systems of common representatives. Such a problem, like many other problems of combinatorial optimization, reduces to the problem of finding an *SDR* for subsets of a finite set.

3.2 Criteria for the Existence of *SDR* a Family of Subsets

With search *SDR* the following questions are solved: is there a complete *SDR* for a given family of subsets of a finite set (a partial *SDR* always exists because subsets are non-empty); how to determine the maximum (by the number of subsets) system of representatives. Further, we assume that $S_1, S_2, ..., S_k$ are subsets of a finite sets *S={1,2,3,…,n}*. If the total number of sets is finite, and the sets themselves are infinite, then we can discard all elements except *n* in each S_i, *i=1,…,k.* The existence of an *SDR* can be established by the following criteria a)-b).

a) The first criterion follows from Hall's theorem. The subsets $S_1, S_2, ..., S_n$ have *SDR* if and only if among the elements of any finite number *k* of sets s_i there are at least s_i distinct elements;

b) Another criterion for the existence of an *SDR* of a family of subsets is based on Sperner's theorem [106].

Definition. The set of subsets $S_1, S_2, ..., S_k$ is called Sperner collection, if none of the sets $S_1, S_2, ..., S_k$ is a part of the other.

Sperner's theorem. Let *S* be a set that consists of n elements, $S_1, S_2, ..., S_k$ be a Sperner collection. Then for the Sperner collection there exists an *SDR*.

For example, sets

S_1 ={1,2,7}, S_2 ={3,5}, S_3 ={2,8}, S_4 ={4,6}, S_5 ={3,7,9},

$$S_6 = \{1,2,3\},\ S_7 = \{1,4,5,8,9\},\ S_8 = 1,6,7\},\ S_9 = \{2,5,8\} \quad (1)$$

are such that neither is part of the other. In this case, for these sets, there exists an *SDR*.

Each of the criteria allows us to conclude that subsets have *SDRs*, but not about their definition. Therefore, such criteria are used, one might say, at the stage of preliminary information processing. The question of searching for a complete or partial *SDR* is solved by special methods of combinatorics or graph theory. Let us consider some of these algorithms for searching for partial or complete *SDR*, preferring to present them in a more natural combinatorial interpretation.

3.3 Algorithms for Constructing *SDR*

The subsets S_i of the set *S* are represented by the matrix *C*. Each row corresponds to one subset. As a demonstration example of the operation of algorithms, we note that for the set (1) the corresponding matrix *C* has the form

$$C=[1\ 2\ 0\ 0\ 0\ 0\ 7\ 0\ 0;\ 0\ 0\ 3\ 0\ 5\ 0\ 0\ 0\ 0;\ 0\ 2\ 0\ 0\ 0\ 0\ 0\ 8\ 0;$$
$$0\ 0\ 0\ 4\ 0\ 6\ 0\ 0\ 0;\ 0\ 0\ 3\ 0\ 0\ 0\ 7\ 0\ 9;\ 1\ 2\ 3\ 0\ 0\ 0\ 0\ 0\ 0; \quad (2)$$
$$1\ 0\ 0\ 4\ 5\ 0\ 0\ 8\ 9;\ 1\ 0\ 0\ 0\ 0\ 6\ 7\ 0\ 0;\ 0\ 2\ 0\ 0\ 5\ 0\ 0\ 8\ 0].$$

In the description of the following search algorithms for *SDR* such a matrix form of representation of subsets S_i, $i=1,\ldots,m$, from a finite set $S=\{1,2,\ldots n\}$ is used, which is more consistent with the formulation of criteria in combinatorial form.

Algorithm 1. Alternate choice of representatives of sets.

The choice of set representatives is carried out in turn, looking at the rows of the matrix *C* from left to right.

The first non-zero element s_{1j} of the first row and *j* of the column will be a representative

of the set S_1. Cross out the first row and column *j*;

b) Go to the next uncrossed line *i*. The first non-zero and not crossed out element of row *i* will be a representative of the set S_i;

c) Repeat step b) as far as possible in order to determine the representatives of the sets. If there is no representative for S_i on some i, $i \le n$, then the search process stops and we get a partial *SDR*.

Such a method in many cases can reveal the full *SDR*. However, it may happen that in the presence of a complete *SDR* for the sets S_i, by Algorithm 1 we can determine only a partial *SDR*. The volume of such a system depends on the order of viewing the sets S_i.

Algorithm 2. Search for SDR. taking into account the number of elements in subsets

1) Find all rows *i* containing only one positive element. Each such element will be a representative of the set S_i, and the corresponding row *i* and column *j* of the matrix C are crossed out;

2) Find all columns *j* containing only one positive element. Each such element from row *i* will be a representative of the set S_i, and the corresponding row *i* and column *j* of the matrix *C* are deleted;

3) If representatives for all sets are determined, then the process ends;

4) If there are uncrossed rows or columns, then the numbers k_i and k_j are determined:

k_i is the number of uncrossed out elements in the *i*-th line, k_j is the number of non-crossed-out elements in the *j*-th column;

5) k_1 and k_2 are defined as the minimum of k_i and k_j, respectively;

6) If $k_1 \leq k_2$, then a row containing k_1 elements is selected, one of the elements is selected, and the row and column are crossed out;

7) If $k_1 > k_2$, then a column containing k_2 elements is selected, one of the elements is highlighted, and the row and column are crossed out.

All selected elements at the end of the algorithm constitute the *SDR*, which can be full or partial. For an example with matrix (2), we get the full *SDR*:

$$s = \{s_{23}, s_{32}, s_{44}, s_{61}, s_{17}, s_{86}, s_{59}, s_{75}, s_{98}\}.$$

Algorithm 2 can be used as an auxiliary algorithm in more complex algorithms that always determine a complete *SDR*. (if such a system exists) or establish that such a system does not exist for given subsets of a finite set.

3.4 Algorithm 3. Search for *SDR* using Hall's theorem

At the beginning of Algorithm 3, Algorithm 1 is applied. The sets $S_1,\ldots,S_n$ are considered to be numbered and fixed in a certain order. The process of searching for representatives is carried out in turn, starting from the first set and up to the set S_k. If *k* is equal to *n*, then we get the full *SDR*. If $k<n$, then this means that all elements of S_{k+1} are representatives of other sets. In such a situation, the *SDR* search process is not stopped. The possibility of the existence of an *SDR* is clarified, in particular, the existence of a representative for the set S_{k+1}. This part of Algorithm 3 requires the algorithmization of the process of further search for *SDR*. A sequence of sets T_0, T_i, $i=1,\ldots,t$ is constructed, where T_0 is the initial set, consisting of elements S_k, T_i is a set consisting of those elements of the set S_p whose representative is an element from the set T_0 and not yet used. The elements of the set T_i are written to the set T_0 after the elements that have already been used. The process of constructing sets T_i continues until some element is found that belongs to some set from $S_1,\ldots,S_k$ with representatives $a_1,\ldots,a_k$. In parallel, a set *Q* is constructed, whose elements are the numbers of the sets from which the elements were added to T_0. With the completion of the construction of the sets T_0 and *Q*, the *SDR* is rebuilt and a transition is made to the consideration of the next subset.

For matrix (2), following Algorithm 3, we obtain representatives *a={1,3,2,4,7}* for the first five sets (represented by the first five rows). For the sixth set, S_6 *={1,2,3}*, it is not possible to find a representative, since all its elements are representatives of other sets. However, in this situation, one can also find a representative for S_6 using the following function included in Algorithm 3. Let us compose the initial set T_0 from the elements of S_6. The corresponding set is *Q={6,6,6}*. We include element *7* from the first set in T_0 and correct *Q*. We get T_0 *={1,2,3,7}* and *Q={6,6,6,1}*. After the second adjustment, T_0 and Q are T_0 *={1,2,3,7,5}* and *Q={6,6,6,1,2}*. Now we rebuild the *SDR*. As a result, a representative is determined for the set S_6, and in

general, for six sets, after the restructuring, we obtain a set of representatives a and a set *P* of the corresponding numbers of the sets S_i, *i=1,2,…,6*, i.e.

$$a = \{3,5,1, 2,4,7\},$$
$$P = \{6,2,1,3,4,5\}.$$

The main part of the *SDR* reforming algorithm 3 is represented using MatLab (Attaway Stormy, 2019). The function is applied with the following data: the set T_0 consists of *t* elements S_{k+1} that have been used as representatives of other sets; *rs=k+1*; *Q* – consists of the numbers of the sets whose elements are included in T_0, $a=(a_1,a_2,\ldots a_k)$ и $P=(P_1,P_2,\ldots P_k)$ are such that a_i is a representative for the set with number P_i, *i=1,2,…,k*; the S=*{S(i1,j1)}* matrix consists of elements, the rows of which represent the sets $S_1,S_2,\ldots S_n$; *prvd* – a sign of the possibility of extending the set of representatives *a* by the representative S_{k+1}.

For the FileSDR.m script, the input is:

rs=6;

k=5; a=[1 3 2 4 7]; P=[1 2 3 4 5]; % Representatives *a* for *k* sets *P*

t=3; T0=[1 2 3]; Q=[6 6 6].

Transformations of the input data and the results obtained are commented:

FileSDR.m

```
for i1=1:rs-1
  prvd=0;
   for j1=1:n s=S(i1,j1);
     if s==0 continue; end
     % Checking whether an element s belongs to the set
     h1=0;
     for k1=1:t
       if s~=T0(k1) h1=h1+1; end
     end
     if h1<t continue; end
     if h1==t h2=0;  % Element s does not belong to
       for k2=1:k
         % Checking if element s matches one from the
         % representatives of the set a.
         if s~=a(k2) h2=h2+1; end
       end
     end
     if h2<k t=t+1;
      % Coincidence of the element s with one of the
      % representatives of a and the inclusion of s in
```

```
            T0=[T0 s]; Q=[Q i1];
          end
             if h2==k
           % s does not match with none of the representatives a
           t=t+1;
           T0=[T0 s]; Q=[Q i1]; prvd=1;
           break;
          end
        end  % j1
        if prvd==1
          % For  sets, there is an SDR in the form (a,P)
          % The vectors ap and Tp are introduced, which are used
          % in the formation of the SDR
          ap(1:k)=1; Tp(1:t)=0;
      xQ=Q(t); yT=T0(t); Tp(t)=1;
          PR=t+k;
          for h=1:PR
           pr=0;
            for k1=k:-1:1
              if xQ==P(k1)
                xp=P(k1); ya=a(k1); ap(k1)=0; pr=1; break;
               end
            end
            if pr==0 break; end
           if pr==1
              for t1=t:-1:1
                if T0(t1)==ya
                  xQ=Q(t1); yT=T0(t1); Tp(t1)=1;
                end
              end
            end
          end   % h
           % Sets ap and Tp are defined
          % and an extended SDR of the form (an, Pn) is formed
          r=0;
          for k1=1:t
            if Tp(k1)==1
              r=r+1; an(r)=T0(k1); Pn(r)=Q(k1);
            end
          end
          for k1=1:k
            if ap(k1)==1
              r=r+1; an(r)=a(k1); Pn(r)=P(k1);
            end
          end

          for k1=1:k
            if ap(k1)==1
              r=r+1; an(r)=a(k1); Pn(r)=P(k1);
```

```
            end
          end
          k=r; P=[Pn]; a=[an];
          fprintf('\n SDR is defined for the sets S1,...,S6:');
          fprintf('\n a='); for ic=1:k fprintf(' %d',a(ic)); end
          fprintf('\n P='); for ic=1:k fprintf(' %d',P(ic)); end
          fprintf('\n');
          break;
        end % prvd==1
      end  % i1
```

As a result of the execution of this script:

>> FileSDR

SDR is defined for the sets $S_1,...,S_6$:

a= 3 5 1 2 4 7

P= 6 2 1 3 4 5

>>

If the set T_0, after the completion of the construction of the sets T_i, will consist of representatives of the sets $S_1,\ldots,S_k$, and $k<n$, we conclude that there is no *SDR* for the sets $S_1,\ldots,S_{k+1}$ due to the absence of a representative for a separate set S_{k+1}. Thus, using Algorithm 3, one can construct a complete *SDR* or establish non-existence of such for given sets S_i, $i=1,2,\ldots n$.

3.5 Algorithm 4. Reducing SDR search to solving an assignment problem

To do this, the matrix *C*, representing the sets S_i with elements from $J=\{1,2,...,n\}$, is replaced by the matrix S,

$$S=\{s_{ij}\},$$

where $s_{ij}=1$, if the *i*-th set S_i contains an element *j* from *J*, otherwise, $s_{ij}=M$, and *M* is a sufficiently large positive number.

We solve the following function minimization problem

$$f=\sum_{i=1}^{n}\sum_{j=1}^{n}s_{ij}x_{ij}$$

provided that only one element x_{ij} is *1*, the remaining elements of the row or column are *0*.

Based on the obtained optimal solution, *SDR* is constructed for sets S_i, *i=1,...,n*. Let us assume that the optimal solution, found, for example, by Kuhn's method [138], has the form

$$x = \left\{ x_{i_1 j_1}, x_{i_2 j_2}, ..., x_{i_{1n} j_n} \right\}.$$

Then the coordinate $x_{i_k j_k}$ is *1* and the corresponding value $s_{i_k j_k}$ is *1* or *M*. If $s_{i_k j_k} = 1$, then the element of the row i_k in the column j_k of the matrix C is considered to be a representative of the set, $s_{i_k j_k} = 1$. All such representatives constitute the maximum possible *SDR*. If $s_{i_k j_k} = M$, then the set S_{i_k} has no representative. The optimal value is *f=k+(n-k)*M* in accordance with the fact that *k* sets will have representatives and *n-k* sets will not have representatives.

% Reducing *SDR* search to solving an assignment problem

The sets are given *S(1)={a,c}, S(2)={a}, S(3)={a}, S(4)={a,b),*
S(5)={a,b,d,e,f}, S(6)={a,b}.

We form a matrix C corresponding to the sets *S(i)*, $i = \overline{1,n}$,

```
n=6;
C=[1 M 1 M M M;  1 M M M M M;  1 M M M M M;
   1 1 M M M M;  1  1  M  1  1   1;   1 1 M M M M];
```

```
function SDR4
 it=0; U(1:n)=0; V(1:n)=0; SUV=0;

 while it<IT
 it=it+1;
 % = === == Step 1 Reduction of matrix C by rows and columns = ====
 % Reduction of C by rows
 mc1=min(C');
 for ic=1:n C(ic,:)=C(ic,:)-mc1(ic); end
 smc1=sum(mc1);
 mc2=min(C);
 for ic=1:n C(:,ic)=C(:,ic)-mc2(ic); end
 % Reduction of C by columns
 smc2=sum(mc2);
 SUV=SUV+smc1+smc2; % Value of minimum elements

 % = == = Step 2 Start searching for the optimal solution = == =
    k=0; is(1:n)=0; js(1:n)=0; mi(1:n)=0; mj(1:n)=0;
    p(1:n)=0; a(1:n)=0; cvn=0; pv=0;
```

```
while pv<n  % Determining independent zeros
    pv=pv+1;
% 1. The only zero of each row is taken as an independent zero
for i=1:n
    if mi(i)>0 continue; end
    cn=0;
    for j=1:n
      if C(i,j)==0 && mj(j)==0 cn=cn+1; j1=j; end
    end
    if cn==1  mj(j1)=j1; mi(i)=i;
      k=k+1; p(k)=i; a(k)=j1;
    end
  end

% 2. The only zero of each column is taken as an independent zero
  for j=1:n
    if mj(j)>0 continue; end
    cn=0;
    for i=1:n
      if C(i,j)==0 && mi(i)==0 cn=cn+1; i1=i; end
    end
    if cn==1 mi(i1)=i1; mj(j)=j;
      k=k+1; p(k)=i1; a(k)=j;
    end
  end

   % ki - number of zeros in line i
  k1=M; r1=0;
  for i=1:n
    if mi(i)>0 continue; end
    ki=0;
    for j=1:n
      if C(i,j)==0 && mj(j)==0 ki=ki+1; end
    end
    if (ki>0 && ki<k1) k1=ki; r1=i; end
  end

   % kj - number of zeros in column j
  k2=M; s1=0;
  for j=1:n
    if mj(j)>0 continue; end
    kj=0;
    for i=1:n
      if C(i,j)==0 && mi(i)==0 kj=kj+1; end
    end
    if (kj>0 && kj<k2) k2=kj; s1=j; end
  end

if r1==0 && s1==0 break; end
```

```
  % An independent zero is selected from a string
  if (k1>0 && k1<=k2) || (k1>0 && k2==M)
     for j=1:n
        if C(r1,j)==0 && mj(j)==0
           j1=j; mj(j1)=j1; mi(r1)=r1;
           k=k+1; p(k)=r1; a(k)=j;
           break;
        end
     end
  end

% An independent zero is selected from a column
  if (k2>0 && k2<k1) || (k2>0 && k1==M)
     for i=1:n
        if C(i,s1)==0 && mi(i)==0
           i1=i; mi(i1)=i1; mj(s1)=s1;
           k=k+1; p(k)=i; a(k)=s1;
           break;
        end
     end
  end
  if k==n break; end
end  % Completing the search for k independent zeros (p,a)

if k==n   % Optimal assignment found
   f=0; cn=0;
   for i=1:k  f=f+c(p(i),a(i));

   end

   for i=1:k
      if c(p(i),a(i))<M cn=cn+1; end
   end
   if cn<n
      fprintf('\n SDR does not exist cn=%d',cn);
   end
   break;
end  % k==n

% Step 3 Determining the minimum set of lines
% that cross out all zeros of the matrix C

mst(1:n)=0; for j=1:n msb(j)=j; end
for i=1:n
   if p(i)>0 mst(p(i))=p(i); end
end
fprintf('\n');
for i=1:n
```

```
    if mst(i)==0 fprintf(' i1=%d',i); end
  end
  for i=1:n fprintf(' %d', mst(i)); end

  kz=0; pv=0;
  while pv<3
     pv=pv+1;

    for i=1:n
      if mst(i)>0 continue; end

      if C(i,j)==0
        pr=0;
        for r=1:k
           if i==p(r) && j==a(r) pr=pr+1;  end
        end
        if pr==0 j1=j; msb(j1)=0;
          for r=1:n fprintf(' %d',msb(r)); end
          break;
        end
      end

    end
  end

  for i=1:n
    if mst(i)==0 continue; end
    for j=1:n

      if C(i,j)==0 && msb(j)==0
         pr=0;
         for r=1:k
            if i==p(r) && j==a(r)
               i1=i; pr=1; mst(i1)=0;
               for r=1:n fprintf(' %d',mst(r)); end
               break;
            end
         end
      end
    end
  end

  end  % while pv<6

  % The minimum set of rows and columns containing all zeros,
 % determined from mst and msb
 kz=0;   % kz – number of lines crossing out all zeros
  if k<n
  % Transform matrix C to obtain new zeros
```

```
mnz=M; i0=0; j0=0;
for i=1:n
   for j=1:n
      if mst(i)==0 && msb(j)>0
        if C(i,j)<=mnz mnz=C(i,j); i0=i; j0=j; end
      end
   end
end
% mnz - minimum unoccupied element

for i=1:n
   for j=1:n
      if mst(i)==0 && msb(j)>0
         C(i,j)=C(i,j)-mnz;
      end
   end
end
   for i=1:n
      for j=1:n
         if mst(i)>0 && msb(j)==0
            C(i,j)=C(i,j)+ mnz;
         end
      end
   end
SUV=SUV+(n-k)*mnz;
end % k<n
end % it
```

3.6 Algorithm 5. *SDR* search associated with the construction of cycles

1) We build the matrix *S* based on the sets $S_1, S_2, \ldots, S_n$;

2) As a representative of the set S_i from the group of sets $S_{i_1 j}$, $S_{i_2 j}, \ldots, S_{i_k j}$ with one element in the row is and in the same column *j*, we take an element from row *i*; delete all lines $S_{i_1 j}$, $S_{i_2 j}, \ldots, S_{i_k j}$;

3) Similarly, we look through columns with one element and as a representative of the set S_j from the group of sets $S_{ij_1}, S_{ij_2}, \ldots, S_{ij_s}$, we take an element from column *j*; cross out all columns $S_{ij_1}, S_{ij_2}, \ldots, S_{ij_s}$;

4) For the remaining elements of the matrix *S* after deleting all rows and all columns with one element , we build a cycle with pairs *(r, s)* of the following form

$$(i_1, j_1),\ (j_1, i_2),\ (i_2, j_2),\ (j_2, i_3), \ldots, (j_r, i_s) \qquad (3)$$

5) From the odd pairs of the constructed cycle, we take for the sets S_r the representative s_{rs}. Cross out all rows of the matrix *S* corresponding to the sets S_r;

6) From the remaining elements of the matrices S, we build another cycle (if any) and select representatives for the remaining rows;

7) Repeat steps 4)-5) until all rows of the matrix *S* are crossed out.

As a result of Algorithm 5, we obtain a system of sets with representatives, as well as sets that do not have representatives, since they do not correspond to the Sperner criterion. In this algorithm, it is possible to select representatives from the cycle and, according to another option, i.e. choice of representatives of sets from even pairs of cycle (3). Thus we get another system of different representatives.

The presented algorithms can be used in solving problems from various fields of research related to the choice of their solution from several alternative options. The algorithms are presented without taking into account the features of the objects that form the initial subsets of the finite set, which ensures their universality. They can be used in transport-type problems, in planning, in the formation of independent expert groups, in resource allocation, and in other studies.

3.7 Method of searching for *SDR* by incidence matrix

There is a way to find several *SDR* using the incidence matrix *D* of the family of sets instead of the matrix *C*. Put one instead of a nonzero element in the matrix *C* and we get *D*.

In general, if a family of subsets $S_1, S_2, ..., S_m$ of a set of elements $a_1, a_2, ..., a_n$ is given, the matrix *D* has the form

$$D = \begin{pmatrix} d_{11} & \dots & d_{1j} & \dots & d_{1n} \\ \dots & \dots & \dots & \dots & \dots \\ d_{i1} & \dots & d_{ij} & \dots & d_{jn} \\ \dots & \dots & \dots & \dots & \dots \\ d_{m1} & \dots & d_{mj} & \dots & d_{mn} \end{pmatrix}$$

where $d_{ij} = 1$ only if $a_j \in S_i$ and $d_{ij} = 0$ otherwise.

An ordered set of elements equal to 1, in which the elements lie in different columns, represents the *SDR* of the incidence matrix *D*. The number of

SDR of a family of sets $S_1, S_2, ..., S_m$ is equal to the number of *SDR* of the incidence matrix of this family.

Algorithm 2 for searching for *SDR* by incidence matrix

The task is to find the *SDR* $d_{1j_1}, d_{2j_2}, ..., d_{mj_m}$ by the incidence matrix *D*. To perform the search, we apply the following transformations.

1. Determine the number of ones by rows, r_i, $i = \overline{1,m}$.

2. Rearrange the rows in matrix *D* in ascending order r_i.

3. Determine the number of ones by columns, s_j, $j = \overline{1,n}$,

4. Rearrange the columns in descending order s_j.

5. Select elements 1 from different rows and columns. First, select 1 from the upper left matrix of dimension 3×3. Then select units from the other three rows and three columns that make up the next matrix most saturated with units, etc. As a result, we can obtain several *SDR*, from the analysis of which an acceptable one is selected.

Example. Let's apply algorithm 2 to matrix *D*:

$$D = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

After the transformation *D*, we obtain the matrix *C*:

$$C = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Selecting elements with value 1 from three-dimensional matrices

$$m_1 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}, \quad m_2 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \quad m_3 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix},$$

we get two *SDR* from m_1, three *SDR* from m_2 and m_3. This means that there are a total of 18 *SDR* corresponding to the permutations $(j_1, j_2, ..., j_9)$:

(1,2,3,4,5,6,7,8,9), (1,2,3,4,5,6,8,7,9), …, (6,5,2,4,3,1,8,7,9), (6,5,2,4,3,1,9,8,7).

3.8 On the extension of the method for solving the choice problem

There are many problems whose formulations fit into the framework of the choice problem. One of such formulations of the choice problem (assignment problem), as a transport-type problem, was given in the introduction. Therefore, it is natural that it can be solved by methods for solving the transport problem.

Let us present one of such methods for solving the traveling salesman problem.

Let *c* be an integer efficiency matrix of the traveling salesman problem of order n.

Algorithm.

1. We perform the reduction of the matrix *c* by rows and columns.

2. We determine the cyclic set of (i_r, j_r), $r = \overline{1,n}$ independent zeros (one from each row and column), i.e. the maximum cyclic partial transversal of the zero elements of the reduced matrix *c*.

3. If $r = n$, then the optimal solution is found.

4. If $r < n$, then we rearrange the rows and columns containing independent zeros to the upper left corner.

5. Among the unoccupied (not crossed out) elements, we find the minimum cyclic element mm in position (*p*, *q*), complementing the partial cyclic transversal.

6. Add the value of $m + h$, (*h*>0, *integer*):

a) to all elements of rows and columns containing unoccupied elements, with the exception of row *p* and column *q*;

b) to all elements of rows containing the zero element in column *q* and to all elements of columns containing the zero element in row *p*.

7. Subtract the minimum element *m* from the unoccupied (not crossed out) elements; add *m* to the occupied (crossed out twice) elements; the rest (crossed out once) remain unchanged.

8. Repeat steps 1-7 until a complete set of independent zeros is obtained and the optimal solution is determined from them.

3.9 Heuristic algorithms

The purpose of this section is to develop and experimentally study a method for solving the traveling salesman problem using a set of heuristics. Complementarity, algorithmic polynomiality and practical significance of the heuristics were the main criteria for including it in the set in order to approach the creation of an accurate method for solving problems of sufficiently large size in an acceptable time.

One of the group of qualitative methods for solving discrete optimization problems are heuristic methods, which are a combination of logical methods and methodological rules of theoretical research and the search for an acceptable solution. Heuristic methods rely to some extent on intuition, experience, guesswork, ingenuity and creative search of specialists when solving economic problems [137]. They are especially useful for solving discrete economic models, since the search for an optimal solution may require a complete enumeration.

In the general case, heuristic algorithms cannot find the optimal (exact) solution. However, applying them to a practical task leads to much better recommendations than an arbitrary and random decision. This applies, for example, to the class of practical discrete problems, in particular, the traveling salesman problem, for the solution of which heuristic techniques are proposed.

Many heuristic techniques are used for both symmetric and non-symmetric matrices.

For greater certainty, let us consider the symmetric case. In this case, the conditions of the traveling salesman problem are often defined by considering a specific communication network, which can be represented as a graph $G = (V, E),$, where V is the set of vertices and E is the set of edges, such as a map of the cities of a country in which the cities constitute the set of vertices and the roads connecting these cities are arcs.

Constructing a distance matrix for the traveling salesman problem can be done in two ways. The first method consists in calculating the matrix of

distances d_{ij} between vertices *i* and *j* according to the rectangular coordinates of the vertices taken from the city map or by geographical coordinates with their conversion into rectangles.

The second way is to calculate the shortest distance between vertices along the actual paths between the points. In this case, the algorithm that solves this problem is the following. For this communication graph, we assume that the distance between points *i* and *j* is equal to d_{ij}^0.

The value of d_{ij}^0 is limited, $d_{ij}^0 < \infty$, if the points are directly connected by elementary arcs, and unlimited, $d_{ij}^0 = \infty$, if it is impossible to go from one point to another, bypassing the other vertices of the graph, $d_{ii}^0 = 0$.

A set of quantities d_{ij}^k, *k=1,2,3*, … , is constructed using the Floyd algorithm or the following rule [2]

$$d_{ij}^k = \min_l (d_{il}^{k-1} + d_{lj}^0), \ \mathrm{k} = 1, 2, 3, ..., \mathrm{r},$$

where *r* is determined from condition $d_{ij}^r = d_{ij}^{r-1}$.

The matrix $\left\{d_{ij}^r\right\}$ is the matrix of the shortest distances between any two points on the network. To describe the properties of the complete graph in which the Hamiltonian path is sought, the zero elements of the matrix $\left\{d_{ij}^r\right\}$ along the main diagonal are replaced by a sufficiently large number.

3.9.1 Decription of heuristics

Using a set of heuristic algorithms leads to routes close to the optimal route. The study of heuristics *E1-E4* for inclusion in such a set is based on their experimental research on a variety of test problems. If the best solution found by the previous heuristics is improved by the new one, then it is included in the set.

Heuristic E1.

Let $\mathrm{c} = \{\mathrm{c}_{ij}\}$ be a matrix that coincides with the original matrix $d = \{d_{ij}\}$ for the traveling salesman problem. For each pair (i_1, j_1), the point number i_1 is entered into set I, the point number j_1 is entered into set J, and then a cyclic admissible solution of the traveling salesman problem is constructed.

1. For a fixed number of column j_1 a row with a number i_2 is selected, for which

$$c_{i_2,j_1} = \min_s c_{s,j_1},$$

where *s* is the line number from *1* to *n*, which does not belong to set I.

2. The number i_2 of the selected line is entered in the set I and in the contour in construction (i_1, j_1, i_2).

3. In matrix *c* element (i_2, j_1) is blocked, so as not to get an incomplete cycle. For this, it is assumed that $c_{i_2 j_1} = M$, where *M* is a fairly large number.

4. With a fixed line number i_2 select the column with number j_2, for which

$$c_{i_2,j_2} = \min_s c_{i_2,s},$$

where s – line number from 1 to n, which does not belong to the set j_2.

5. The number j_2 of the selected column is inserted into set J in the outline in construction (i_1, j_1, i_2, j_2).

6. It is assumed that the c_{j_2,i_2} element of the *c* matrix is equal to a sufficiently large number to eliminate the possibility of the (j_2, i_2) pair entering the contour in the construction. The number j_2 is inserted into a set *J*.

7. The number $j = j_2$ is fixed and go to step 1. Next, $j = j_2$, is assumed and calculations continue with step 2.

Steps 1-7 are repeated until a complete cyclic solution is obtained.

In order to obtain a large number of improvement solutions and statistically evaluate the occurrences of arcs (i, j) in the optimal solution within the heuristic *E1*, the transformation of the original matrix is used and obtaining a working matrix *c* with the elements

$$\mathrm{c}_{ij} = d_{ij} + p_i + q_j,$$

where p_i and q_j – uniformly randomly distributed numbers from the range (A, B) chosen by the researcher. Such a transformation is possible, because the optimal solution of the traveling salesman problem with matrices $\{\mathrm{c}_{ij}\}$ and $\{d_{ij}\}$ coincides.

From the cyclic solutions obtained, the solution corresponding to the minimum length is selected. The best solution obtained by the first heuristic *E1* is the initial one to obtain new improved solutions using the following heuristics *E2-E4*.

Heuristic E2. An improvement of the initial solution can be obtained by changing the places of two elements.

Let i and j be the numbers of two elements of the solution, $i < j$,

$$U = (1, 2, ..., i-1, i, i+1, ..., j-1, j, j+1, ..., n, 1)$$

The new solution has the form

$$V = (1, 2, ..., i-1, j, i+1, ..., j-1, i, j+1, ..., n, 1)$$

The difference between the route lengths is

$$\Delta_{ij} = d_{i-1,i} + d_{i,i+1} + d_{j-1,j} + d_{j,j+1} - d_{i-1,\mathrm{j}} - d_{j,i+1} - d_{j-1,\mathrm{i}} - d_{i,j+1}$$

With a positive Δ_{ij} value, U is replaced by a V and with solution V the next improved solution is sought.

The improvement solutions found are used for a statistical evaluation of the affiliation of the transitions (from point i to point j) to the optimal route.

Using test examples, it was found that an improvement in the solution is also possible after passing the second heuristic using the following third heuristic.

Heuristic E3. Element k is inserted between the other two.

The cyclic decision in the construction based on the initial solution U starts with the choice of the first transition, the first pair of points *(i,j),* but the remaining numbers of points in quantity *n-2* will be entered according to the following scheme. For the first pair of points, a partial route is selected from three options

$$(k, i, j),\ (i, k, j) \text{ and } (i, j, k).$$

Suppose a partial route (i_1, i_2, i_3) of minimum length has been selected. In the following third heuristic step, partial paths are searched:

$$(k, i_1, i_2, i_3),\ (i_1, k, i_2, i_3),\ (i_1, i_2, k, i_3) \text{ и } (i_1, i_2, i_3, k) \qquad (1)$$

The best of the four partial routes is selected. Similarly, the calculation process continues until a full $(i_1, i_2, i_3, ..., \mathrm{i}_{n-2}, i_n)$ cyclic route is obtained as the best of the partial routes constructed by analogy with (1).

Heuristic E4. Improving the admissible solution using the inversion operation.

One of the heuristic algorithms can be like this. We choose an arbitrary Hamiltonian contour and replace the two adjacent vertices in it. If the path value falls, then the previous path is remembered and move to the next pair of

points. If the route has improved, then a new circuit is stored and treated in analog fashion.

For a problem with symmetric conditions, one such algorithm is as follows. We take a (more or less random) contour $i_1, i_2, ..., i_n, i_1$, that passes through all points of the graph. We renumber the points of the graph again, so that in the analyzed contour they meet in sequence $1, 2, ..., n$. For two arbitrary points *i* and *j*, $i < j$, , of this graph, an operation called inversion is introduced.

We write the original route in the following form

$$1, 2, ..., i-1, i, i+1, ..., i+k-1, i+k, j, j+1, j+2, ..., n-1, n, 1 \qquad (2)$$

The inversion consists of the following. We exclude arcs *(i,i+1)* and *(j,,j+1)* from contour (2). Instead, arcs *(i, j)* and *(i + 1, j + 1)* are included in the contour and the entire path between *i + 1* and *j*, goes in the opposite direction. As a result of such an operation, a new outline is obtained

$$1, 2, ..., i-1, i, i+1, j ..., i+k, i+k-1, ..., i+1, j+1, j+2, ..., n-1, n, 1 \qquad (3)$$

The difference between the lengths of contours (2) and (3) is equal to

$$\Delta_{ij} = d_{i,i+1} + d_{j,j+1} - d_{i,j} - d_{i+1,j+1}.$$

With a positive Δ_{ij}, it is possible to improve the contour (2).

Example of inversion shown in Fig.1. Algorithmically, the solution (3) is obtained as follows as a function of *i* and *j*. We denote the initial solution (2) by *U*. An improving solution V is obtained using the following transformation:



Figure 1. Contour a) until inversion 1,2,3,4,5,6,7,8,1;
b) after inversion of 1,2,3,7,6,5,4,8,1.

1) It is believed $V = U$;

2) When *i+1=j* the solution *V* does not change. The inversion in this case does not lead to a change to a change in *U;*

3) When $i+2=j$ the replacements are produced $V_{i+1}=U_j$, $V_j=U_{i+1}$;

4) In other cases, substitutions are made $V_k=U_{j+i+1-k}$, $k=i+1,i+2,...,j$.

In the new contour (3), in a completely similar way, a favorable reversal can be sought. To study the practical possibilities of the heuristic, computational experiments were performed, in particular, the following.

3.9.2 The experimental part

Experiments were performed with arrays of different sizes $n<=100,$, which showed the possibility of using the software developed by the author to solve the metric problem of the traveling salesman. For example, for a problem with $n=100,$the solution is shown in Fig. 2. To determine the solution, it is necessary to specify the (x,y) coordinates of the graph points:

x=(20, 22, 26.5, 24.5, 27.5, 36, 40, 44, 36.5, 34, 24.5, 29, 29, 26, 22, 14, 9, 3, 4.5, 7, 10, 4, 13, 12.5, 17, 11.5, 12, 19, 19.5 22, 21, 23, 26.5, 28.5, 29, 57, 23, 28.5, 33, 36.5, 45.5, 45.5, 46, 50, 50, 54.5, 51, 52, 55, 55, 66, 64, 63, 64, 63, 64.5, 62, 64, 62, 59, 58, 53, 50, 47, 42, 45, 45, 48.5, 51, 59, 66.5, 64, 60, 55, 56, 61, 64, 65, 66.5, 58.5, 50, 44, 41, 34, 35, 32, 26, 23.5, 23, 19, 14, 12, 11, 7, 6, 6, 1.5, 2.5, 15, 20),

y=(19, 16, 21, 23.5, 23, 22, 23, 22, 38, 46, 29, 37, 33, 32.5, 28.5, 34, 32.5, 36.5, 39, 47, 49, 55, 65, 56.5, 55, 45.5, 43.5, 48, 52, 53, 58.5, 58.5, 56, 57.5, 62, 62, 67, 66.5, 63, 68, 62, 57.5, 54, 51, 54, 60.5, 61.5, 64, 64, 67.5, 62.5, 60.5, 60, 58.5, 56, 54, 52, 49, 43, 43, 37, 45, 47, 42, 41.5, 40, 36, 38, 32.5, 24, 24, 18, 22.5, 16.5, 13.5, 14, 12, 11, 10, 10, 6, 15, 10, 13.5, 7, 8, 8.5, 5, 7, 8, 4, 3, 8, 9, 11, 18, 18, 21, 24, 19)

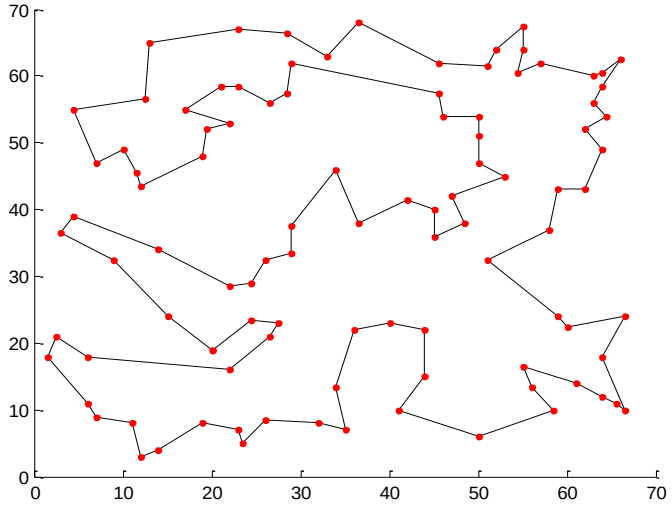


Figure 2. Detour of 100 points

An approximate solution is obtained

M=(67, 66, 65, 9, 10, 12, 13, 14, 11, 15, 16, 19, 18, 17, 99,,100, 1, 4, 5, 3, 2, 96, 98, 97, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84, 6, 7, 8, 82, 83, 81, 80, 75, 74, 76, 77, 78, 79, 72, 71, 73, 70, 69, 61, 60, 59, 58, 57, 56, 55, 54, 51, 52, 53, 36, 46, 49, 50, 48, 47, 41, 40, 39, 38, 37),

by which its length is calculated.

The following example with data based on a map of the cities of Moldova. It is necessary to present a route that passes only once through each of the 36 cities of Moldova with a minimum length. In Table 1, the numbers from 1 to 36 indicate the cities: Chisinau, Balti, Comrat, Orhei, etc.

Table 1

1 Briceni	10 Floresti	19 Nisporeni	28 Leova
2 Ocnita	11 Singereni	20 Straseni	29 Cimislia
3 Edinet	12 Rezina	21 Dubasari	30 Causeni
4 Donduseni	13 Soldanesti	22 Cruileni	31 Cantemir
5 Riscani	14 Rezina	23 Chisinu	32 Comrat
6 Drochia	15 Telenesti	24 Ialoveni	33 Basarabeasca
7 Soroca	16 Ungeni	25 Hincesti	34 Stefan Voda
8 Glodeni	17 Calarasi	26 Anenii Noi	35 Cahul
9 Balti	18 Orhei	27 Tighina	36 Taraclia

An acceptable cyclic solution *M* determined using the heuristics *E1-E4* is:

M=(1, 3, 5, 8, 11, 9, 12, 15, 17, 16, 19, 20, 23, 24, 25, 28, 31, 35, 36, 32, 29, 33, 34, 30, 27, 26, 22, 21, 18, 14, 13, 10, 7, 6, 4, 2, 1).

If you start with the first city, 1-Briceni, it is recommended to bypass the cities in sequence 3 - Edinet, 5-Riscani, 8-Glodeni, etc. Supported city numbers are shown in Fig. 3.

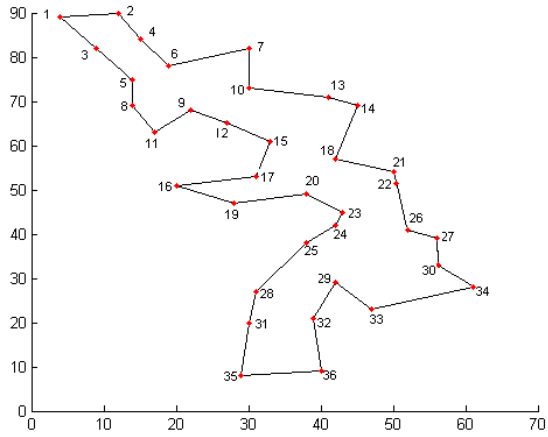


Figure 3. Bypass route of Moldovan cities

The accuracy of the approximate algorithm for solving the traveling salesman problem is investigated, for example, using a series of specially generated tests of varying complexity. The resulting set of acceptable and improving solutions represent useful statistics that are processed.The processing results are represented by an *n*-dimensional array $H = \{h_{ij}\}$, where h_{ij} is the frequency of occurrence of the element *(k,1)* in the improved solutions generated by heuristics *E1-E4*. Based on the matrix *H* we can make an assumption (heuristic or hypothesis) that its element of *(k,1)* with the highest frequency indicates the entry of the transition from one point *k* to another *l* with the distance d_{kl} in the optimal solution. So the matrix d_{kl} can be used to develop new heuristics.

For the considered example, it was proved that the arcs of the cyclic solution *M* obtained by four heuristics *E1-E4* are included in the set of arcs corresponding to the non-zero elements of the matrix *H*. The number of zero elements of the matrix *H* is much less than zero. This information can be used to reduce the size of the original matrix and find the best solution.

Experiments have shown that for problems with a symmetric tariff matrix for moving from one point to another of medium size, *n* <= 100, practically good solutions can be obtained.

3.10 Statistical method of approximate solution of the discrete optimization problem

1. One of the areas lying in the field of quantitative methods in economics is the problem of choice. A large number of situations that arise in various fields of activity are associated with the solution of choice problems as a problem of discrete optimization. Here are some examples of such tasks.

Let's say there are *n* devices and *n* products. Each product can be processed on any device. The processing time of one part on one device may differ from the processing time on another. The processing times are considered to be known for each product and on each of the devices. All this information is set in the form of an n by n table, where at the intersection of row *i* and column *j* is the processing time of item *i* on device *j*. The task is to organize the production of products, i.e. place them on devices in such a way that the estimate of the total operating time is the smallest. Another example. A fairly large number of items are being considered. For each of the items, its weight is known. It is necessary to answer the question, is it possible to divide this set into two parts of equal weight?

The given tasks have characteristic properties: the finiteness of a variety of choices; each option is compared with a quantitative characteristic (total operating time, weight of items in each of the two parts, etc.). It is required to choose a variant with the property that its numerical characteristic satisfies a certain previously formulated condition or to answer the question of the existence of such a variant. It is often required that this characteristic reach an extreme value on the chosen option. Such problems are related to discrete optimization problems.

The most obvious method for solving discrete optimization problems (with a finite number of options) is to consider all feasible options and select the required one. Difficulties in solving such problems arise when the number of options is large enough. With an increase in the number and variety of such selection problems, problems began to arise of the existence of a common methodology for their solution, assessing the quality of an approximate solution. Questions arose about organizing an effective search in order to make available the solution of problems with a sufficiently large number of options, i.e. how to reduce the solution of a problem with a large dimension to a set of similar ones, but of a lesser dimension.

All selection tasks can be divided into two groups. Problems for which there are effective algorithms for solving. For them, the complexity of the

solution is expressed by a polynomial function of the dimension of the problem. Another group consists of problems for which similar algorithms are not yet known. One of these problems is the traveling salesman problem, which serves as a testing ground for the development, research and testing of new methods and therefore is of great theoretical and practical interest.

2. About the statement of the problem. For the development of a statistical algorithm, the main requirements are a cyclic closed traversal of n cities with minimal costs. Perhaps the costs mean travel time, ticket price, distance traveled, or an arbitrary non-negative number. The statement of the problem uses a cost matrix with diagonal elements equal to zero or a sufficiently large number, depending on the calculation stage.

3. Solution methods. The traveling salesman problem is NP-hard, and there is no algorithm yet that guarantees its solution in time polynomial in the number of cities n. However, for a small number of cities, there are many solution methods, including exact ones. Exact methods not only find some solution, but also prove at the end of their work that this is the best solution. An example of such a method is a complete enumeration of permutations of n-1 numbers (since the starting city is fixed) and choosing the best route in accordance with them. This approach is practically difficult to implement for large values of n. Directed backtracking is an enumeration of options "around" some solution with the cutting off of paths that have a length greater than the best path to the current moment.

The branch and bound method is the most effective known method of cutting off "unpromising" nodes, due to the analysis of the distance matrix. When searching for the optimal solution, a binary tree is built (means that at each node 2 branches are generated, which are interpreted in the sense that the traveling salesman goes to some city or does not go to it). Linear programming is used to minimize (with constraints) the linear form of the $c \cdot x$, where x is the required binary vector of dimension $n(n-1)/2$, the components of which x_{ij} are equal to 1 or 0, depending on whether the (i, j)-th edge enters the path or not. Matrix c (of the same dimension), whose elements are equal to the lengths of the edges.

Heuristic methods are usually much faster than exact ones. however, they do not guarantee optimality of the found solution [58]. The result of their combination can then be used as a first approximation for further improvement, for example, using backtracking search. Among these methods - the greedy algorithm, when choosing the next city, takes the nearest city that has not been visited before; shoelace method - a geometric variation of the greedy algorithm in

which it is assumed that cities are surrounded by a closed circuit. It gradually stretches out, trying to pass through all the cities, increasing its length in minimal ways; a sliding search is performed, as a result of which cities from a small part of the path are swapped. Then this the "window of options" slides along the entire path. The method has various variations and turns out to be an effective way to improve the solution found by the previous two heuristic methods.

3. Probabilistic methods actually make a certain number of iterations, making random changes in the path, in anticipation of obtaining a shorter one. Examples of such methods are the annealing method in which the permutations of cities take place with a gradually "fading" intensity. At the same time, the best solution found is always retained. The genetic algorithm is a more "advanced" version, which creates a large number of different paths and which are constantly changing, they break apart and merge with each other, and at the same time exchanging individual sections.

Characterizing the methods of solving traveling salesman problems that exist today, the following feature can be noted (as a hypothesis): for each developed algorithm, it is possible to define such a traveling salesman problem (to construct such a cost matrix) that will not be solved (an optimal route will not be obtained). For example, for the "greedy" algorithm - an algorithm for finding the minimum distance by choosing the shortest, not yet chosen edge, provided that it does not form a cycle with already chosen edges. This algorithm is called "greedy" because at the last steps it is necessary to choose the longest edge.

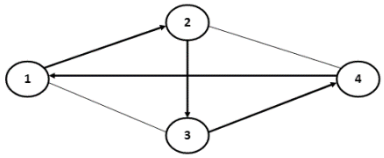


Figure 1. A network representing a narrow rhombus,
and the tour found by the greedy algorithm

Let's see how the greedy algorithm behaves when solving the TSP. Here it will turn into a strategy "go to the nearest (which has not yet entered) city". Consider, for example, the network in Fig. 1 representing a narrow rhombus. Let the traveling salesman start from city *1*. The “go to the nearest city” algorithm will bring it to city *2*, then *3*, then *4*; at the last step, you have to go along the long diagonal of the rhombus, from *4* to *1*. As a result, you will get not the shortest, but the longest round *(1,2,3,4,1)*.

Widely available online programs solve the traveling salesman problem (up to several dozen cities), with the exception of special cases leading to the solution of traveling salesman problems sometimes up to the size of several thousand cities, but without a guarantee of obtaining an exact solution, i.e. in the answer we can get some approximate solution.

A feature of this problem is that with an increase in the number of cities, the total number of different combinations of paths grows strongly (exponentially). And at the same time, the time for calculating the result obtained by the applied algorithm also increases. Therefore, the main thing in optimization can be reduced to discarding obviously not minimal paths during computations. It is necessary to specify a criterion that would cut off unnecessary branches in the shortest path search tree.

Methods for solving the Euclidean traveling salesman problem should provide for a procedure for eliminating paths that have intersections. Such paths are not optimal. Indeed, let there be some path shown below in Figure 2.

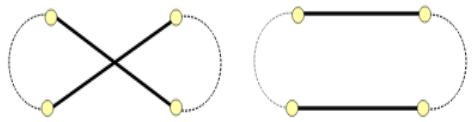


Figure 2. 2-opt conversion example

If in the first picture we move two sections of the path (as in the second picture), then the total path will become shorter (the sum of the diagonals of a *4*-gon is always longer than the sum of two opposite sides):

Interesting questions of a statistical nature are connected with the Euclidean traveling salesman problem. Consider, for example, a square in which *n* cities are randomly distributed with a uniform distribution. It turns out that in most cases such "maps of cities" have shortest paths of approximately the same length, proportional to the square root of *n*. Results of this kind can be obtained by methods with which you can quickly obtain the optimal route. As a rule, the exact solution of the gearbox is achieved for special tasks. At the same time, such methods are required for general problems, which would allow a wider variety of experiments to be carried out. This explains the appearance of works by many researchers on discrete optimization, including the traveling salesman problem. As a result, there is an intensive accumulation of experience in solving the PC, which can lead to the emergence of new, effective methods of solving them.

4. Algorithm for statistical solution of discrete optimization problems

The algorithm is based on the use of statistics of the frequencies of transitions from one point to another. The first part of the route is formed by the maximum frequencies of transitions. The other part of the route, additional to the cyclic permutation, is found after removing the rows and columns in which the transitions from the first part are located.

The sample consists of cyclic routes that are generated according to the scheme considered below.

The generation of the solutions that make up the sample is presented (in terms of the Vogel method for drawing up the basic plan of the transport problem). The corresponding step and the result of this step are shown by an example:

Departure points	Destinations				Stocks
	B1	*B2*	*B3*	*B4*	
A1	*M*	*4*	*7*	*8*	*1*
A2	*4*	*M*	*3*	*7*	*1*
A3	*7*	*3*	*M*	*5*	*1*
A4	*2*	*8*	*5*	*M*	*1*
Needs	*1*	*1*	*1*	*1*	

In the table, along the main diagonal of the matrix of elements c_{ij}, there are numbers *M* large enough to provide a transition from two different points *i* and *j* (except for the starting point and the final point in the route, which coincide).

The first reference plan is found by the approximation method, taking into account the features of the problem.

1) For each line A_i, find the difference between the two minimum tariffs written in this line,

$$U1=(3,1,2,3),$$

and put them in the corresponding additional column.

2) For each column B_j, we find the difference between the two minimum tariffs written in this column,

V1=(2,1,2,2),

and put them in the corresponding additional line.

3) Let us choose the largest difference from all the obtained. In this case, the largest difference is *3*. In this row, the minimum tariff is *4* and is located at the intersection of runoff A_1 and column B_2. The element A_4 and B_2 also have a difference of *3*, but the element c_{41} is less than c_{12}. Therefore, we fill the cell with c_{41}, setting $x_{41}=1$.

4) Now, taking into account the peculiarities of the problem, we exclude both row *4* and column *1* at the same time.

5) For the rest of the matrix, we carry out similar calculations according to items 1) -4). We get the following two additional columns

U2=(3,4,2, -)

U3=(4,-, 2, -)

and two additional lines

V1=(-,1,4, 2)

V1=(-,1, -, 3),

where the "-" sign means the absence of a value in the corresponding column (row). As a result of calculations, we obtain the plan *(4,1), (1,2), (2,3), (3,4),* which turned out to be cyclical. But this is not always the case. In the general case, this scheme may not lead to a cyclical plan, which is required by the conditions of the problem. Therefore, we modify the procedure in order to generate cyclic plans in the general case, as well as for selecting the best and collecting statistics on the frequencies of occurrence of elements in improving plans.

Algorithm.

1) For each pair (i_1, j_1), $i_1=1,\ldots,n$; $j_1=1,\ldots,n$, we calculate the estimates for the column j_1 and according to its maximum estimate of the difference between the two minimum tariffs, we determine the row i_2. Then, by row i_2, we similarly define the column j_2, and with it the element in the cell (i_2, j_2) to be entered into the plan. So in *n-1* steps we will get a cyclic plan

$(i_1, j_1),(i_2, j_2),\ldots,(i_n, j_n)$ we will receive such plans in the number *n(n-1)*, from which the improving route values are selected for calculations according to the next item 2).

2) We form the matrix *p* from the frequencies pij to include the element *(i,j)* in the cyclic design.

3) We select elements *(i,j)* with the highest frequency p_{ij} from all rows (columns) to be included in the plan.

4) With respect to the reduced data matrix, similar calculations are carried out as in items 1) -3). The resulting parts of the cyclical plan are combined into one complete cyclical design with n elements. If, after the reduction, we have reached a matrix up to a sufficiently small dimension *kxk*, then we add another *k* elements of the cyclic plan under construction using a complete enumeration of possible options.

5. Computational experiments were carried out in order to estimate the number of elements introduced into the route by successive iterations for the investigated approach in solving discrete optimization problems.

The formation of statistical samples in the algorithm is carried out with changing the distance matrix c to the matrix $c^{'}$, adding values p_i to the *i*-th row and q_j *j*-th column. Thus, the new matrix generated by the transformation

$$c^1 = \left\{c^1(i, j)\right\} = \left\{c(i, j) + p_i + q_j\right\},$$

where $p = (p_1, p_2, ..., p_n)$ and $q = (q_1, q_2, ..., q_n)$ are random vectors, without changing the optimal solution.

For the solved problems, according to iterations of the selection of elements at the maximum frequencies to enter the best plan, it turned out that at the next iteration at least *50*% of the elements in the route under construction are included. The unification of the two parts of the route under construction was carried out according to operations similar to those shown in Fig. 2. In particular, if one part of the route has the form $(i_1,i_2,\ldots,i_k)$, and the other $(j_{k+1}, j_{k+2},\ldots, j_s)$, then the combination of the parts results in a permutation of the following type of

$$\begin{pmatrix} 1\,2\ldots k\;k+1 k+2\ldots s \\ i_1\,i_2\ldots i_k\;\; j_{k+1}\;\; j_{k+2}\;\ldots j_s \end{pmatrix}$$

provided that $i_k = k+1$. The parts to be joined can always be prepared for such a combination.

To eliminate the effect that the reference plans obtained by the approximation method (used in solving transport problems and in the branch-and-bound method for solving various discrete optimization problems) basic plans at the final stage receive sufficiently large values, in the algorithm when implementing item 4), the values of *k* did not exceed four. This allows you to include in the plan elements with a low frequency, but improving the solution.

3.11 Decomposition of information relative to bypass points by convex hulls

The paragraph presents the results of an algorithmic study of a special optimization problem of the transport type (traveling salesman problem): 1) To solve the problem, a new natural algorithm has been developed, based on the decomposition of the initial data into convex hulls, which has a number of advantages: it is applicable for a fairly large dimension, does not require a large amount of memory, and has fairly good performance. The relevance of the algorithm lies in the fact that in practice programs for problems with the number of traversal points of no more than twenty are widely used. For large-scale problems, the availability of algorithms and programs of this kind is difficult. The proposed algorithm is natural because the optimal solution found by the exact algorithm is not always feasible due to the presence of many other factors that may require some additional restrictions. 2) Another inverse problem solved here is to describe a class of traveling salesman problems that have a predetermined optimal solution. The constructed algorithm 2 allows us to characterize the structure of traveling salesman problems, as well as construct test problems to evaluate the effectiveness of algorithms and other purposes.

One of the problems in the research of transport-type operations (the traveling salesman problem), which has been intensively studied, is to create a closed route that passes through each of *N* points only once and is the least in terms of travel time or cost. The traveling salesman problem (*TSP*) belongs to the class of *NP*-hard problems. Therefore, today, approximate methods (directed enumeration, heuristic, genetic, using neural networks, etc.) are being developed to solve it. The emergence of new solution methods allows for a deeper understanding of the *TSP* problem, which will lead to an effective, accurate solution with a complexity acceptable for practice.

It is known that problems of sufficiently large dimensions have been solved when N reaches several tens of thousands of points. If we are talking

about a specific problem, then by transforming the posed problem and applying one or more known methods, as a rule, we can obtain an optimal solution.

3.11.1 On the solution of special traveling salesman problems.

In the general case, for some classes of problems, it is possible to present an algorithm for obtaining an exact solution. For example, in cases where the finite set of initial points is a convex hull, or close in structure to the set of points of an integer square lattice on the plane.

In the first case, the value of the optimal solution will be equal to the sum of the sides of the polygon corresponding to the convex hull. In the second case, the algorithm for constructing the optimal solution can be described as follows. The entire set of points is decomposed into convex hulls of $P_1, P_2, ..., P_k$ .The first convex hull P_1 is constructed from the set of all initial points. The vertices of the constructed convex hull P_i are excluded from consideration when constructing the next one P_{i+1}. Any point of the initial data will belong to only one of the constructed convex hulls

The solution to the traveling salesman problemm is obtained by the inverse process of combining the vertices of the convex hulls of the P_i, $i = 1, 2, ..., k$. For this, the following procedure is applied. We consider the initial route of the traveling salesman to be a sequential bypass, in one direction, of the vertices of the set $M = P_1$. For two points (a_1, b_1) and (a_2, b_2) from M and two points (a_3, b_3) and (a_4, b_4) from $P_{i+1}, i = 1, 2, ..., k-1$, all six distances between these points are calculated and analyzed, and thus two points from the second convex hull are included between two adjacent points from M. Note that such a procedure for including points from the lower shell to the upper one can also be applied in more general situations of mutual arrangement of points and can be used in the corresponding algorithms.

3.11.2 On the solution of the *TSP* using the data decomposition algorithm for convex hulls

Let's consider a more complicated case of location of initial data on a plane. Let us present in more detail the corresponding algorithm for constructing convex hulls and a demonstration example.

Example 1. Given points (xdat, ydat). Number of points n=30.

xdat=[2 5 10 15 17 16 12 6 3 3 8 13 15 16 14 10 5 5 8 10
13 14 14 12 8 8 9 12 13 10];
ydat=[11 13 13 11 7 3 1 2 5 10 12 11 9 6 3 2 5 7 10 11
10 7 5 3 4 6 9 8 5 5].

The result of constructing convex hulls is shown in Fig. 2.

First, the outer convex hull is determined, on the boundary and inside of which all the data is located.

The points located on the shell $k+1$ are added to the points on the shell k. The solution is obtained as a sequence of points obtained as a result of the union of all constructed convex hulls $1, 2,..,k$. Depending on the location of the points, different criteria apply for inserting a point V from the $k+1$th convex hull between two points A and B on the path obtained after combining the previous k convex hulls.

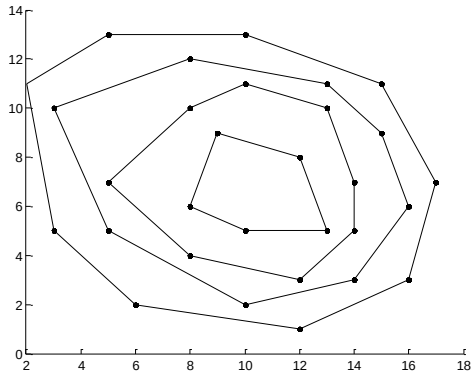


Figure 1. Representation of the initial data by a set of convex hulls

Algorithm 1.

The algorithm 1 for constructing a set of convex hulls and constructing a solution to the traveling salesman problem can be represented by the scheme:

1) Data analysis, identification and elimination of duplicate data;

2) Checking the linearity of the data, i.e. location of points on a straight line;

3) If all points lie on one straight line, then the end of the search for convex hulls;

4) Construction of the convex hull of points. Based on the *x* and *y* coordinates, we determine the numbers of the vertices located on the given convex hull;

5) Eliminate from further consideration a set of points on the constructed convex hull, as well as intermediate points between the vertices of this convex hull:

6) Check if there are points left to determine the next convex hull;

7) If "yes", then go to step 2);

8) After the completion of the process of constructing all shells, starting from the first, outermost, to the last, the number and coordinates of all convex shells are stored. As in the previous case, we begin the reverse process of combining convex hulls. The first and second convex shells are combined according to the criterion of proximity of the vertices of the second to the faces of the first one or by the minimum distance of a given point on the second shell from two points on the first one, etc.;

9) The result of combining convex hulls is a sequence of points, which we take as a "practically" optimal route. As in the previous case, we begin the reverse process of combining convex hulls.

Applying the described algorithm, for the example under consideration, the solution is represented by a sequence of points 1,2,19, 27, 11, 3, ..., 9, 18, 10 in Fig. 2.

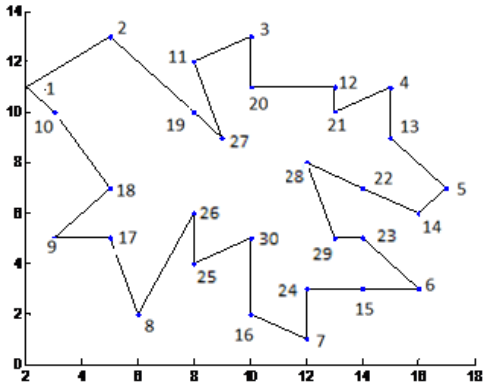


Figure 2. Sequence of bypassing items

Regarding Algorithm 1 and the program implemented on this basis in the Appendix, we note that they represent a demonstration of a new natural approach to solving the symmetric traveling salesman problem. At the same time, computational experiments with traveling salesman problems with uniformly distributed detour points, as in the examples (Fig. 2), show the real possibility of using the developments.

```
% Demonstration program for solving the symmetric traveling salesman problem
        function S_TSP
        clc
        clear all
        rand('state', sum(100*clock))
        global i2 vo PVO xdat ydat i1 m  ctpvo it km kb r

        fprintf('\n Demo results ');
        % Enter: n - number of points, M - a sufficiently large positive number,
        % eps - a sufficiently small positive number
        % Enter point coordinates (xdat,ydat)
        fprintf('\n  Initial data n=%d \n (xdat,ydat): ', n);
        for k=1:n fprintf(' (%d,%d)', xdat(k), ydat(k)); end

         figure
         hold on
         for i=1:n
            plot(xdat,ydat,'r.')
            text(xdat(i),ydat(i), num2str(i));
         end

         np=n; ndat=n; idat=1:n;
         ndat=n; x=xdat; y=ydat;
         ind=1:n; indt=1:n; indo=1:n;

         % Determine the number of repeating points
         i1=0; re=zeros(1,1);
         for i=1:n-1
            for j=i+1:n
              if abs(x(i)-x(j))+abs(y(i)-y(j))<eps
                 i1=i1+1; re(i1)=j;
              end
            end
         end

         if i1>0  % elimination of duplicate points
           x(re)=[]; y(re)=[]; ind(re)=[];
           n=n-i1;
         end

         it=0;
         while n>=3
           it=it+1;
           % Start building CH for (x,y) with ind numbers
         r=0;
         x1=x(1); x2=x(n);
         y1=y(1); y2=y(n);
           for k=2:n-1
              x3=x(k); y3=y(k);
```

```
      a=x3-x1; b=y2-y1; c=x2-x1; d=y3-y1;
      if abs(a*b-c*d)<eps r=r+1; end
    end

  if r==(n-2)    % All points are on the same line
    for k=1:n-1
      x1=x(k); x2=x(k+1); % (x,y) - на прямой
      y1=y(k); y2=y(k+1);
      plot([x1 x2],[y1 y2],'r');
    end
    it=it-1; break;
  end

  kvert=convhull(x,y);  % kvert - CH vertex numbers
  cvo=length(kvert);    % number of vertices on CH

  xkv=x(kvert); ykv=y(kvert);
  xkv(cvo)=xkv(1); ykv(cvo)=ykv(1);
  indv=indt(kvert); indv(cvo)=indv(1);

  ctvo(it)=cvo;
  for k=1:cvo VO(it,k)=indv(k); end
   % CH vertices at iteration it

  for k=2:cvo
    x1=xkv(k-1); x2=xkv(k);  % (xkv,ykv) - coordinates of CH vertices
    y1=ykv(k-1); y2=ykv(k);
    plot([x1 x2],[y1 y2],'r'); %
  end  % k

  xp=x; yp=y;
  xp(kvert)=[]; yp(kvert)=[];   % (xp,yp) - points without vertices CH
  rx=length(xp);  ind=1:rx; indt(kvert)=[];
  % The remaining points along with intermediate ones on the edges CH
  promej=zeros(n,n); cpt=zeros(1,n); ocpt=0;
  for j=1:rx     % Is the point intermediate
    for i=2:cvo
      x1=xkv(i-1); x2=xkv(i);
      y1=ykv(i-1); y2=ykv(i);
      c1=x2-x1; c2=y2-y1;
      c3=xp(j)-x1; c4=yp(j)-y1;
      c5=c3*c2-c4*c1;
      if abs(c5)<eps
        cpt(i-1)=cpt(i-1)+1; ocpt=ocpt+1;
        promej(i-1,cpt(i-1))=indt(j);
      end
    end   % i
  end  % j
 if ocpt>0
```

```
  indp=zeros(1,ocpt);
  for i=1:cvo
     upr=zeros(1,cpt(i)); x1=xkv(i); y1=ykv(i);
     for j=1:cpt(i) i1=promej(i,j); x2=xdat(i1); y2=ydat(i1);
        upr(j)=sqrt((x1-x2)^2+(y1-y2)^2);
     end
     [b,c]=sort(upr);
     for j=1:cpt(i) g(j)=promej(i,c(j)); end
     for j=1:cpt(i) promej(i,j)=g(j); end
  end  %  i
end

  r=0;
  for i=1:cvo-1
    r=r+1; indp(r)=indv(i);
    for j=1:cpt(i) r=r+1; indp(r)=promej(i,j); end
  end

  ctpvo(it)=r;
  for j=1:r PVO(it,j)=indp(j);
  end
  k=n-r;

  r=0;
  x=zeros(1,k); y=zeros(1,k); ind=zeros(1,n); indt=zeros(1,n);
  indo(indp)=-1; ind=1:k;

  for i=1:ndat
     if indo(i)>0
        r=r+1; x(r)=xdat(i); y(r)=ydat(i); indt(r)=indo(i);
     end
  end
  n=k;
end

% Built by it CH
% Numbers and coordinates of the remaining points:indt(i),x(i),y(i)) i=1:n

 prlvo=0;   % prlvo - sign of linearity of the last CH
 if n>0
   it=it+1; prlvo=1;
   for i=1:n VO(it,i)=indt(i); PVO(it,i)=indt(i); end
   ctvo(it)=n+1;  ctpvo(it)=n;
   VO(it,n+1)=VO(it,1);
 end

 % Building a route based on data from two adjacent convex hulls
 i1=ctpvo(1);
 m=zeros(1,i1);
```

```
for i=1:i1 m(i)=PVO(1,i); end
m(i1+1)=m(1); i1=i1+1;
for i=1:i1 fprintf(' %d', m(i)); end

for vo=2:it
  i2=ctpvo(vo);
  zahvat();
  % Joining points from the inner CH to the outer one
  fprintf('\n Current route vo=%d i1=%d m: ',vo,i1);
  for i=1:i1 fprintf(' %d', m(i)); end

  figure;
  hold on
  for i=1:i1-1
    j1=m(i); j2=m(i+1);
    ai=xdat(j1); bi=ydat(j1);
    aj=xdat(j2); bj=ydat(j2);
    plot([ai aj],[bi bj],'k'); % External CH
    plot(ai, bi,'r.');
  end

  PVO(vo,i2+1)=PVO(vo,1);
  for i=1:i2
    j1=PVO(vo,i); j2=PVO(vo,i+1);
    ai=xdat(j1); bi=ydat(j1);
    aj=xdat(j2); bj=ydat(j2);
    plot([ai aj],[bi bj],'g');  % Attachable CH
    plot(ai, bi,'r.');
    fprintf(' %d', j1);
  end
  hold off
end

fprintf('\n ');
figure;
hold on
f=0;
for i=1:i1-1
   j1=m(i); j2=m(i+1);
   ai=xdat(j1); bi=ydat(j1);
   aj=xdat(j2); bj=ydat(j2);
   f=f+sqrt((ai-aj)^2+(bi-bj)^2);
   plot([ai aj],[bi bj],'r');
   plot(ai, bi,'b.');
end
hold off
fprintf('\n Route length m: f=%6.1f \n ', f);
for i=1:i1-1 fprintf(' %d', m(i));
text(xdat(i),ydat(i), num2str(i));
```

```
 end
 fprintf('\n\n');

 for i=1:it
    ctpvo(i)=ctpvo(i)+1;  PVO(i,ctpvo(i))=PVO(i,1);
 end

 % Getting started with obhod()
 kz=M;  m=zeros(1,ndat); r=0;
 kb=zeros(1,it); km=zeros(1,it);
 if prlvo==1 it=it-1; end
 for vo=1:it-1
   obhod();
 end
if prlvo==1
  % Build route after obhod() and then
  % join the last CH line to the previous CHs
end

function obhod()
  global i2 vo PVO xdat ydat i1 ctpvo km kb kz rkb rkm
  i2=ctpvo(vo+1)-1;
  dmin=1000.0;
  for k=1:i2
    if i2<2 break; end
    i31=PVO(vo+1,k); i32=PVO(vo+1,k+1);
    ak=xdat(i31); bk=ydat(i31);
    ak2=xdat(i32); bk2=ydat(i32);
    i1=ctpvo(vo)-1;
   for i=1:i1
     if i==kz-1 | i==kz | i==kz+1 continue; end
     i4=PVO(vo,i); i5=PVO(vo,i+1);
     ai4=xdat(i4); bi4=ydat(i4);
     ai5=xdat(i5); bi5=ydat(i5);

     dki4=sqrt((ak-ai4)^2+(bk-bi4)^2);
     dk2i5=sqrt((ak2-ai5)^2+(bk2-bi5)^2);
     di4i5=sqrt((ai4-ai5)^2+(bi4-bi5)^2);
     dkk2=sqrt((ak-ak2)^2+(bk-bk2)^2);
     a=dki4+dk2i5; b=di4i5+dkk2;
     dt=a-b;     % dt=min(dki4,dki5);
     if (dmin>dt)
       dmin=dt; kb(vo)=i; km(vo)=k;
       rkb(vo)=PVO(vo,i); rkm(vo)=PVO(vo+1,k);
     end
   end % i
  end % k

  kz=km(vo);
```

```
function zahvat()
  global i2 vo PVO xdat ydat i1 m
  for k=1:i2
    dmin=1000.0;
    i3=PVO(vo,k);
    ak=xdat(i3); bk=ydat(i3);
    for i=1:i1-1
      i4=m(i); i5=m(i+1);
      ai4=xdat(i4); bi4=ydat(i4); ai5=xdat(i5); bi5=ydat(i5);
      dki4=sqrt((ak-ai4)^2+(bk-bi4)^2);
      dki5=sqrt((ak-ai5)^2+(bk-bi5)^2);
      di4i5=sqrt((ai4-ai5)^2+(bi4-bi5)^2);
      dt=dki4+dki5-di4i5;
      % dt=min(dki4,dki5);
      if (dmin>dt)
        dmin=dt; imin=i; im=i4; ib=i5;
      end
    end  % i

    for j=i1:-1:imin+1 m(j+1)=m(j); end
        m(imin+1)=PVO(vo,k);
    i1=i1+1; m(i1)=m(1);
  end % k
```

Demo results

Initial data n=30

(xdat,ydat): (2,11) (5,13) (10,13) (15,11) (17,7) (16,3) (12,1) (6,2) (3,5) (3,10) (8,12)
(13,11) (15,9) (16,6) (14,3) (10,2) (5,5) (5,7) (8,10) (10,11) (13,10) (14,7)
(14,5) (12,3) (8,4) (8,6) (9,9) (12,8) (13,5) (10,5)

Current route vo=1 i1=10 m: 9 8 7 6 5 4 3 2 1 9

Current route vo=2 i1=18 m: 9 17 8 16 7 15 6 14 5 13 4 12 3 11 2 1 10 9

Current route vo=3 i1=26 m: 9 17 8 25 16 7 24 15 6 23 22 14 5 13 4 21
12 20 3 11 19 2 1 10 18 9

Current route vo=4 i1=31 m: 9 17 8 26 25 30 16 7 24 15 6 23 29 28 22 14 5
13 4 21 12 20 3 11 27 19 2 1 10 18 9

Route length m: f= 73.6
9 17 8 26 25 30 16 7 24 15 6 23 29 28 22 14 5
13 4 21 12 20 3 11 27 19 2 1 10 18

CHAPTER 4. A NATURAL METHOD FOR SOLVING THE TRAVELING SALESMAN PROBLEM

4.1 An approach to solve the traveling salesman problem with help of the potentials method

The exposition of this paragraph follows article [52]. Approximate solution of the traveling salesman problem is carried out by modifying the potentials method to solve the transport problem and using the necessary optimality condition. In the process solutions, the cost matrix is converted to eliminate unwanted journeys from one point to another.

When using quantitative methods to solve economic problems, it is necessary to take into account the limits that they possess. Transportation tasks are such classes, for example, the selection problem, the maximum flow problem, the shortest path problem, the distribution of products between enterprises, the distribution of cultivated area, the optimization of the supply of raw materials and materials for manufacturing enterprises, the optimization of goods deliveries from warehouses to retail stores, optimizing passenger traffic. This allows the development of more efficient methods for solving them compared to general methods.

For the transportation problem, efficient solution algorithms have been developed. Here, the theory of solving the transportation problem is essentially used to solve the traveling salesman problem, taking into account its limitations.

We adapt technology for solving the transportation problem to determine the optimality of an admissible solution to the traveling salesman problem. The initial data of the *TSP* is represented by the matrix of distances (costs)

$$d = \begin{pmatrix} d_{11} & d_{12} & \dots & d_{1n} \\ d_{21} & d_{22} & \dots & d_{2n} \\ \dots & \dots & \dots. & \dots \\ d_{n1} & d_{n2} & \dots & d_{nn} \end{pmatrix}$$

Going from point *i* to point *j*, the traveling agent suffers loss d_{ij}. The cost of travel through the points $i_1, i_2, ..., i_n$ is equal to

$$f = \sum_{k=1}^{n-1} d_{i_k j_{k+1}} + d_{i_n i_1}.$$

Enter the initial data of the traveling salesman problem, represented (in transportation problem terminology) as the supplier inventory vector $a = (a_1, a_{2,}, ..., a_n)$, the consumer query vector $b = (\mathrm{b}_1, \mathrm{b}_{2,}, ..., \mathrm{b}_n),$, where

$$a_i = 1 + \varepsilon,\ i = \overline{1, n},$$

$$b_i = 1,\ i = \overline{1, n-1},\ b_n = 1 + n\varepsilon,$$

ε – quite a small number.

The solution to the problem of the traveling salesman is done in the following way:

1) Formation of a valid cyclic solution. For this purpose, different ways of generating admissible solutions are used. We will identify each variable x_{ij} with the route *(i,j)*. Finding a non-degenerate base solution is equivalent to finding at most *2n–1* routes (out of the n^2 possible ones). The routes will be organized in a table similar to the one in which the problem data is organized, with each route corresponding to a box *(i,j)*.

To find some basic initial solutions there are a multitude of methods (the north-west method; the line minimum method; the column minimum method; the minimum cost method; the maximum difference method; modified Vogel method). Some methods try not only to find the plan, but even to find the best one. In this plan, we will present the Vogel method, adapted to solve the traveling salesman problem: a) for each line and each column, the difference between the two lowest costs of its routes is calculated (the difference can be 0 if the minimum is multiple) and find the maximum between these differences;

b) among all the routes on the lines and columns corresponding to this maximum, the minimum cost route is chosen (if the minimum is multiple, one is taken at random). The maximum possible is transported on this route. This

maximum is equal to the minimum between the quantity still available at the supplier corresponding to this route and the quantity still needed at the consumer corresponding to the route, at the time of choosing this route. It is transported in this way in order to use as few routes as possible and therefore to obtain a basic solution. The adaptation of the Vogel method means that at each stage of the formation of an admissible $\{x_{ij}\}$ solution the requirement of its cyclicity is taken into account.

Another approach to forming a feasible solution $\{x_{ij}\}$ is possible as in the following table

				$1+\varepsilon$
1				ε
	1			ε
		$1-\varepsilon$		2ε
		ε	1	

unde ε – număr destul de mic, where ε – quite small number, $\varepsilon>0$.

2) We determine the potentials u_i and v_j;

3) Further we will calculate

$$s_{ij} = u_i + v_j - d_{ij}$$

for all zero values x_{ij};

4) We build the next feasible solution. Two cases are considered: a) among all s_{ij} estimates there is none positive; b) some s_{ij} estimates are positive.

In case a) the plan is optimal. According to the optimal plan (for the transportation problem constructed in the last step) a permutation is formed

$$(\text{xv}, \text{yv}) = \begin{pmatrix} i_1 i_2 \ldots i_n \\ 1\ 2\ \ldots n \end{pmatrix}$$

where i_k (the number k of the upper part *xv* of the permutation (*xv*,*yv*)) means the number of the row in matrix $\{x_{ij}\}$ where element $x_{i_k j}$ is close to one. If the permutation (*xv*,*yv*) is cyclic, then we obtain an admissible solution to the traveling salesman problem. If the permutation (*xv*,*yv*) is not cyclic, then it is decomposed into cycles and for each cycle of type $j_1\ j_2 \ldots j_s$ (of length *s*) the cost matrix *d* is corrected, namely, the element $d_{j_s j_1}$ is replaced by a sufficiently large number *M* to prohibits the passage from point j_s to point j_s.

In case b), for each positive s_{ij} evaluation corresponding to the zero element of the x_{ij}, a closed cycle is constructed (by the general rules): starting from this route, passing only through the routes of the solution, going only vertically or horizontally and each passage from a route to another being only perpendicular to the previous passage. There is only one circuit with these properties that passes through an even number of routes. Starting with "+" from the route that will enter the base, the circuit routes are alternately noted with "+" and "–". It is marked with θ the minimum of the quantities transported on the routes marked with "–" and the route for which this minimum was obtained is the one that will leave the base. θ is subtracted from the quantities transported on the routes marked with "–" and added to those marked with "+", the routes that are not on the circuit keeping their value.

Then, we construct the next admissible solution. The computation continues according to the transportation problem algorithm. The study of the possibility of forming an acceptable cyclic approximation to each positive s_{ij} (for zero x_{ij}) is a specific preliminary stage of the considered approach to solving the traveling salesman problem.

5) From the admissible constructed cyclic solutions, the best one is chosen.

Experiment. Experiments were conducted on a number of test problems. Admissible solutions were obtained which were compared with those found using the author's programs [52].

Conclusions. The approach proposed here for solving *TSP* is promising for problems with large different dimensions and can serve as a basis for developing its modifications and experimental research to efficiently solve discrete optimization problems.

This section presents for the first time the idea of improving the technology of the potentials method for solving the traveling salesman problem. The following sections develop this idea. As a result, original algorithms and programs for solving the traveling salesman problem were developed.

4.2 Methods for constructing cyclic solutions

Schemes for constructing an initial base plan for an approximate solution of the traveling salesman problem using the method of potentials are given. The requirements for the initial solution are provided (cyclicity, balance of "reserves" and "needs", non-degeneracy). The procedures for the formation of cycles for the transition to a new plan and the calculation of potentials are simplified, which expands the possibilities for an effective approximate solution of problems of a sufficiently large dimension.

The search for a solution to the traveling salesman problem using the potential method was carried out using several methods for constructing the initial reference plan of the corresponding transport problem with a certain constraint. The peculiarity of the construction is that "reserves" and "requirements" are equal to one and to overcome the degeneracy and cyclicity of the plan, special approaches are required, among which the following were used.

Scheme 1. Consists of two parts. In the first part 1.1 in the plan X under construction (in the beginning, a square zero matrix of dimension n), identity elements are determined. In the second part 1.2, $n-1$ more elements are introduced into the plan X to eliminate the degeneracy and one more element (fictitious zero), which makes it possible to simplify the procedure for forming cycles for the zero elements of the plan X and obtaining a new non-degenerate support plan.

1.1. Inserting X elements $X(i,j)$ into the base plan under construction, the values of which are equal to 1, which means the transition from point i to point j in the traveling salesman route.

A permutation of $p=(i_1,i_2,...,i_n)$ is generated, where i_k is the number of the kth bypass point. The cyclic traversal of points is represented by a set of pairs

$$(x,y)=((x_1,y_1),(x_2,y_2),...,(x_n,y_n)),$$

where $x_i=p_{i+1}$ if $i<n$ and $x_i=p_1$ if $i=n$; $y_i=p_i$, $i=1,2,...,n$.

$$X(x_i,y_i)=1.$$

1.2. Elimination of degeneracy of the base plan by adding positive elements ε (is a rather small number, for example $\varepsilon=0.01$) in base plan X,

For $i=1,2,...,n$. and $i<n$

$$X(x_{i+1},y_i)=\varepsilon,\quad xe_i=x_{i+1},$$

if $i=n$

$$X(x_1,y_n)=\varepsilon,\quad xe_i=x_1.$$

Taking into account the fact that $ye_i=y_i$ for all i, a pairs of (xe_i,ye_i) are formed, in order to analyze the quality of the constructed solution in all schemes (the number of non-zero elements in the matrix X, the number of elements in rows and columns, the sums in rows and columns are counted), as well as checking the non-match condition xe_i and ye_i, since the peculiarity of the problem under consideration is the transition from one point to another once and only once.

Scheme 2. To be able to extend plan X independently of part 1.1, in the second part 1.2, elements are added using a function to generate uniformly distributed random values (for example, the rand() functions in C/C++). In each column (taking into account the fictitious zero) an element equal to eps is inserted. As a result, in each column and in each row there will be one unit and one element ε.

Scheme 3. From the generated permutation $p=(i_1,i_2,...,i_n)$ the positions of elements equal to one are determined in the original plan X, i.e. couples (x_i,y_i), $i=1,2,...,n$ (by analogy with 1.1).

Further from permutation $q=(i_n,i_{n-1},...,i_1)$, in which the elements are in reverse order than in p, the second group of elements is formed in positions (xe_i,ye_i), $i=1,2,...,n$, for base plan X:

$$xe_i=\begin{cases}q_{i+1}, & i<n\\ q_1, & i=n\end{cases}$$

$$ye_i = q_i .$$

$$X(x_i, y_i) = 1, \tag{1}$$

$$X(xe_i, ye_i) = eps. \tag{2}$$

Scheme 4. To analyze the properties of cycles for zero elements of the reference plan, the matrix *X* (feasible solution) is determined in accordance with the adjusted expressions (1) and (2).

We will show the construction of a base plan using the example of the matrix *X* for *n=5*, i.e. dimensions *5x5*, for which, by the positions of elements close to one, it is possible to construct a permutation (bypass route of points),

$$X = \begin{pmatrix} 0 & 0 & 0 & 0 & 1+\varepsilon \\ 1 & 0 & 0 & 0 & \varepsilon \\ 0 & 1 & 0 & 0 & \varepsilon \\ 0 & 0 & 1-\varepsilon & 0 & 2\varepsilon \\ 0 & 0 & \varepsilon & 1 & 0 \end{pmatrix}$$

The possibility of obtaining new cycles can be increased by changing the position of the nth column and rows numbered *n* and *n-1*. In accordance with such a transformation of *X*, it is necessary to transform the distance matrix corresponding to the task.

Scheme 5. After determining all units based on the generated permutation *p*, as in (1), additional elements of the matrix *X* are inserted into the first unoccupied element of the column, starting from the first. To insert missing additional elements in column *j1* and row *i1*, a pair is defined in the sequence *(xe(i),ye(i))*, for which

$$X(xe(i), ye(i)) = \varepsilon, \;\; X\left(xe\left(i\right), j1\right) = 0 \;\text{ and }\; X\left(i1, ye\left(i\right)\right) = 0 .$$

Next is the replacement

$$X(xe(i), ye(i)) = 0,$$

$$X(xe(i), j1) = \varepsilon,$$

$$X(i1, ye(i)) = \varepsilon$$

If there is no additional element in row *i1* and column *j1* and $X(i1, j1) = 0,$ then a replacement is made

$$X(i1, j1) = \varepsilon.$$

Scheme 6. The initial base plan X of the traveling salesman problem is constructed by modifying the Vogel method, which is known from practice as one of the effective ways to build a baseline plan for a transport task. In other words,, the method of forming a permutation in clause 1.1 of scheme 1 is changed, and additional elements can be inserted into the initial plan under construction according to one of the options of clause 1.2 in schemes 1-5. In more detail, this scheme, which was used in the work of the [49-62], is represented by the following code.

```
void ModVogel()
{
   /*  For every pair (ic,jc), ic is not equal to jc, a permutation
is being built
   for(ic=1; ic<=n; ic++) {
      for(jc=1; jc<=n; jc++) {
         if (jc==ic) continue;
        plan(ic,jc);
        }
    }
}

void plan(ic,jc)
{
   /*  c_ij - distance matrix between points
       Pairs (x_i, y_i) determine the position of units in the
matrix X
    */
   for(i2=1; i2<=n; i2++) {
     xv[i2]=0; yv[i2]=0; mrs[i2]=0;
   }
   i=ic; j=jc;
    /*    (mrs[r],mrs[r+1]) – transition from point mrs[r] to
point mrs[r+1]  */
   r=1; mrs[r]=i; r++; mrs[r]=j;
   xv[i]=1; yv[j]=1; it=i; jt=j;
   i=j;
   /*  The stroc() and stolb() functions are executed until the
```

```
full definition of the cyclic
                route, which is determined by the vector mrs
            */
            stroc();  /* Determining column j by row i */
            stolb();  /* Determining row i by column j */
            mrs[n+1]=mrs[1];
        }
        void stroc()
        {
           for(k=1; k<=2; k++) {
/* The difference mul of two minimal elements is determine
   stl[2][j]  and stl[1][j]  in column j of the distance matrix c_ij
*/
                stl[k][j]=M; rt=M;
                for(i2=1; i2<=n; i2++) {
                    if (xv[i2]>0 || yv[i2]>0) continue;
                     if (k==1 && c[i2][j]<rt) {
                            rt=c[i2][j]; istl[k][j]=i2;
                          }
                    if (k==2 && c[i2][j]<rt && i2!=istl[1][j]) {
                            rt=c[i2][j];  istl[k][j]=i2;
                          }
                }
                     stl[k][j]=rt;
            }
                mu1=stl[2][j]-stl[1][j];
                r++; mrs[p]=istl[1][j]; i=istl[1][j];
                xv[i]=1; yv[j]=1;  it=mrs[r-1]; jt=mrs[r];
                c[j][i]=M;
          i=j;
         }
void stolb()  /*  The line i  is fixed and column j is determined */
        {
            for(k=1; k<=2; k++) {
                str[k][i]=M; rt=M;
                for(j2=1; j2<=n; j2++) {
                    if (yv[j2]>0 || xv[j2]>0) continue;
                 if (k==1 && c[i][j2]<rt) {
                            rt=c[i][j2];  istr[k][i]=j2;
                         }
                 if (k==2 && c[i][j2]<rt && j2!=istr[1][i]) {
                            rt=c[i][j2]; istr[k][i]=j2;
                 }
                 str[k][i]=rt;
```

```
        }
            mu1=str[2][i]-str[1][i];
            r++; mrs[r]=istr[1][i]; j=istr[1][i];
    xv[i]=1; yv[j]=1; it=mrs[r-1]; jt=mrs[r];
            c[j][i]=M;
            j=i;
}
```

Let us give an example of the use of schemes 1-6. Let's assume that the square matrix of distances of dimension $n=9$ has the form:

$$c = \begin{pmatrix} M & 18 & 40 & 27 & 15 & 4 & 13 & 38 & 15 \\ 18 & M & 33 & 9 & 19 & 26 & 18 & 8 & 35 \\ 38 & 33 & M & 17 & 22 & 14 & 26 & 22 & 11 \\ 25 & 10 & 15 & M & 33 & 22 & 6 & 20 & 5 \\ 15 & 21 & 21 & 31 & M & 10 & 26 & 33 & 27 \\ 6 & 27 & 16 & 24 & 10 & M & 22 & 25 & 32 \\ 12 & 19 & 26 & 5 & 25 & 21 & M & 28 & 20 \\ 36 & 7 & 24 & 21 & 31 & 27 & 26 & M & 13 \\ 15 & 33 & 10 & 5 & 27 & 32 & 19 & 12 & M \end{pmatrix}$$

Using, for example, scheme 5, units are inserted according to the generated permutation and additional elements, equal to ε, in each column to the place, which is determined by a random number from *1* to *n-2*. We obtain the matrix *X* (expression (3) on the left), in which there is no additional element in row 8 and column 3. Therefore, at the intersection we make the substitution $X(8,3) = \varepsilon$. Thus, the final expression for *X* (in (3) on the right), completes the construction of the base plan, in which one of the additional elements, equal to ε, is considered fictitious and is not taken into account in the calculations of potentials and in the construction of the cycle.

Schemes 1-6 satisfy the requirements for the base plan (the balance of "reserves" and "needs" is ensured, no verification of the degeneracy of the base plan is required). In addition, such schemes for generating a reference plan make it possible to simplify the stages of calculations using the method

potentials. For example, in schemes that give a basic plan of the form (3), due to the presence in each row and each column of one unit and one additional element ε (there are *n* in total), any one of which can be considered a fictitious zero, which makes it possible to simplify the procedure for calculating the potentials and building a cycle for recalculating the base plan. If potentials are calculated, it is recommended to start with the element of the first row equal to one, then go along the column to the additional element, from it along the next line to one, etc. until all potentials are determined. A similar procedure can be

$$X = \begin{pmatrix} 0 & 1 & 0 & \varepsilon & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & \varepsilon & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \varepsilon \\ 0 & 0 & 0 & 0 & \varepsilon & 0 & 1 & 0 & 0 \\ \varepsilon & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \varepsilon & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & \varepsilon & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \varepsilon & 0 \end{pmatrix}$$

$$\rightarrow \quad X = \begin{pmatrix} 0 & 1 & 0 & \varepsilon & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & \varepsilon & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \varepsilon \\ 0 & 0 & 0 & 0 & \varepsilon & 0 & 1 & 0 & 0 \\ \varepsilon & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \varepsilon & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & \varepsilon & 0 & 0 & 0 \\ 0 & 0 & \varepsilon & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \varepsilon & 0 \end{pmatrix} \qquad (3)$$

used to construct a cycle corresponding to a fictitious or zero off-diagonal element. According to the matrix *X* in (3), starting, for example, from the element (6,4), we obtain the cycle *(6,4), (8,4), (8,3), (9,3), (9,8), (3,8), (3,9), (5,9), (5,1), (7,1), (7,6), (2,6), (2,7), (4,7), (4,5), (6,5).*

The possibilities of generating cyclic solutions by schemes 1-6 increase, if the original distance matrix c is transformed into the matrix $c_{ij}^{'} = c_{ij} + p_i + q_j$

(without changing the value of the optimal solution), where p_i and q_j – random positive numbers, $i, j = \overline{1,n}$.

4.2.1 Construction of initial feasible cyclic solutions based on the cost matrix and the evaluation matrix

One of the most important moments in the development of algorithms for solving discrete and combinatorial optimization problems is the choice of the initial solution. For example, for the traveling salesman problem, it is generally recommended to choose a solution as close to the optimal as possible. For this purpose, it is useful to form an initial cyclic solution based on the cost matrix *mc* and the evaluation matrix *mo*.

Formation of a cyclic solution based on the cost matrix *mc*.

1. We construct a sequence of

mc(1,1), ...mc(1,n), mc(2,1),...,mc(2,n), mc(n,1),...,mc(n,n) (1)

2. By arranging (1) in ascending order, we obtain the sequence *y*.

3. We select ny – the maximum number of elements in the sequence (1) that are not greater than a given sufficiently large number M, max *mc(i,j)<M* for $i, j = \overline{1,n}$.

4. For each element of the $y_k = mc\left(iy_k, jy_k\right)$, *k=1,2,...,ny*, we construct a cyclic solution

$$\left(iy_k, jy_k\right),\ \left(jy_k, jy_{k1}\right),\ \left(jy_{k1}, jy_{k2}\right),\ \left(jy_{k2}, jy_{k3}\right),\ldots,$$

for which element $y_{ks+1} = mc(iy_{ks+1}, jy_{ks+1}) = mc(jy_{ks}, jy_{ks+1})$ is selected as minimal from sequence *y*, after excluding previous elements.

5. The best cyclic solution is selected after passing its evaluation for each *k*.

```
% Generating a cyclic solution usiung the matrix mc
        c=mc;
       [f,xv,yv]=Algoritm(n,c);
function [fmin,xm,ym]=Algoritm(n,c)
    [y,iy,jy,ny]=FMDR_1P(n,c);
    [fmin,xm,ym]=FMDR_4P(n,c,ny,y,iy,jy);

function [y,iy,jy,ny]=FMDR_1P(n,c)
    s=0;
     for i=1:n
        for j=1:n
           if i~=j s=s+1;
              t(s)=c(i,j); it(s)=i; jt(s)=j;
           end
```

```
      end
    end
    [y,k]=sort(t);
    iy=it(k); jy=jt(k);
    ny=length(y);

[fmin,xm,ym]=FMDR_4P(n,c,ny,y,iy,jy)
fmin=Inf; xm(1:n)=0; ym(1:n)=0;
for i=1:ny
   k=1; xn(k)=iy(i); yn(k)=jy(i);
   for j1=2:n
   for j=1:ny
      x1=iy(j); y1=jy(j); kn=0;
      if x1==yn(k)
        for i1=1:k
          if x1~=xn(i1) && y1~=yn(i1) kn=kn+1; end
        end
        if k<n kn1=0; r=0;
          for i1=1:k
            if y1~=xn(i1) kn1=kn1+1; end
          end
          if kn1==k r=1; end
        end
        if kn==k && r==1
          k=k+1; xn(k)=x1; yn(k)=y1; break;
        end
      end  % if x1==yn(k)
    end  % j
   end   % j1
   if k==n-1
     k=k+1; xn(k)=yn(n-1); yn(k)=xn(1);
     f=CelFunc(n,c,xn,yn);   % Calculation of the objective function
     if f<fmin fmin=f; xm=[xn]; ym=[yn]; end
   end
end  % i
```

Generating cyclic solutions based on the *mo* evaluation matrix.

1. We construct a sequence of

$$mo(1,1),\ \ldots mo(1,n),\ mo(2,1),\ldots,mo(2,n),\ mo(n,1),\ldots,mo(n,n) \qquad (2)$$

2. By arranging (2) in descending order, we obtain the sequence *y*.

3. We select ny – the maximum number of elements in the sequence (2), not less than *M*, min *mc(i,j)*>-*M* for $i, j = \overline{1,n}$.

4. For each element $y_k = mo\left(iy_k, jy_k\right)$, *k=1,2,...,ny*, we construct a cyclic solution

$$\left(iy_k, jy_k\right),\ \left(jy_k, jy_{k1}\right),\ \left(jy_{k1}, jy_{k2}\right),\ \left(jy_{k2}, jy_{k3}\right),\ldots$$

for which the element $y_{ks+1} = mo(iy_{ks+1}, jy_{ks+1}) = mo(jy_{ks}, jy_{ks+1})$ is selected as the maximum from the sequence *y*, after excluding the previous elements.

5. The best cyclic solution is selected after passing its evaluation for each *k*.

4.2.2 On the procedure for constructing orthogonal cyclic permutations and its application

To apply the method of potentials in solving the transport type (choice problem, traveling salesman problem) [137-143], procedures for obtaining orthogonal cyclic permutations are applied. Let's consider one such procedure. Let $p = (p_1, p_{2,...,} p_n)$ a cyclic permutation in which p_i, $i = \overline{1,n}$, one of the numbers in the set $N = \{1, 2, ..., n\}$.

Definition. We call a cyclic permutation orthogonal to *p* if all pairs (p_i, q_i), $i = \overline{1,n}$, are distinct.

The procedure for constructing several orthogonal cyclic permutations [3]–[4] becomes much more complicated as their number grows. In practice, when solving problems of transport type with restrictions

$$\sum_{i=1}^{n} x_{ij} = 1, \ j = 1, 2, ..., n,$$

$$\sum_{j=1}^{n} x_{ij} = 1, \ i = 1, 2, ..., n.$$

by the method of potentials, the construction of two orthogonal cyclic permutations is used.

Algorithm 1**.** Generation of random orthogonal cyclic permutations.

Step 1.

1.1. We take an arbitrary random permutation of $p = (p_1, p_{2,...,} p_n)$ using, for example, the randperm(n) function in the MatLab package [142];

1.2. We build a permutation $q = (q_{i_1}, q_{i_2}, ..., q_{i_n})$ orthogonal to *p*. The permutation q is such that all pairs (p_i, q_i), $i = 1, 2, \ldots, n$, are distinct. This is ensured by the fact that for p_1, we take any q_{i_1} not equal to p_1 and i_1 from N.

Then we determine q_{i_2}, not equal to p_2, but i_2 from the remaining n-1 values N. We get a pair of (p_2, q_{i_2}).

1.3. We continue the process for $i = 3,4,...,n$. We get a cyclic permutation

$$\begin{pmatrix} p_1\ p_2 \dots p_n \\ q_1\ q_2 \dots q_n \end{pmatrix} \quad (1)$$

Next, we define a random permutation orthogonal to permutation (1).

Step 2.

2.1. From the numbers $i = 1,2,...,n$ and $j = 1,2,...,n$ we choose two random numbers r and s, such that r does not coincide with one and is not equal to p_1, and from them we determine the first pair, denoted as $\begin{pmatrix} r_{j_1} \\ s_{j_1} \end{pmatrix}$

2.2. Next couple $\begin{pmatrix} r_{j_2} \\ s_{j_2} \end{pmatrix}$ determined so that it does not coincide with $\begin{pmatrix} r_{j_1} \\ s_{j_1} \end{pmatrix}$, where r_{j_2} - random from remaining in N, and $s_{j_1} = s_{j_2}$.

2.3. Other couples $\begin{pmatrix} r_{j_k} \\ s_{j_k} \end{pmatrix}$, $k = 3,4,...,n-3$ are defined so that they do not coincide with the previous ones, and

$$s_{j_k} = s_{j_{k+1}} \text{ for odd k and}$$
$$r_{j_k} = r_{j_{k+1}} \text{ for even k,}$$

and r_{j_k} is not equal s_{j_k}.

After building the first n-3 pairs

$$\begin{pmatrix} r_{j_1}\ r_{j_2} \dots r_{j_{n-3}} \\ q_{j_1}\ q_{j_2} \dots q_{j_{n-3}} \end{pmatrix} \quad (2)$$

it is necessary to complete permutation (2) for the last three indices up to a complete permutation of order n. Let's start the description of the permutation complement procedure with an example.

Example. Let us assume that the matrix x is constructed in accordance with (1) and (2), i.e. non-zero (identity) elements, correspond to permutation (1), and fictitious zeros, which are considered as sufficiently small positive numbers, correspond to permutation (2). Diagonal elements that do not

participate in the formation of permutations (1) and (2) are conventionally denoted by the letter M.

$$x=\begin{pmatrix} M & 0 & \varepsilon & 1 & 0 & 0 & 0 \\ 0 & M & 0 & \varepsilon & 1 & 0 & 0 \\ 1 & \varepsilon & M & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & M & 0 & 1 & 0 \\ \varepsilon & 0 & 0 & 0 & M & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & M & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & M \end{pmatrix}$$

We cross out rows 4, 6 and 7, as well as columns 5, 6 and 7, in which there is no fictitious zero.For the remaining elements, we compose the matrix m:

$$m=\begin{pmatrix} 0 & 1 & 0 \\ 0 & M & 0 \\ 0 & 0 & M \end{pmatrix}$$

By the matrix m, we complete the permutation (2). There is one option for complementing the permutation (2): (2,1), (3,2) and (1,3).

In the general case, for any n, options 1)-6) are formed according to the three-dimensional matrix:

$$1)\ a_{11}\ a_{22}\ a_{33};\quad 2)\ a_{11}\ a_{32}\ a_{23};\quad 3)\ a_{21}\ a_{12}\ a_{33};$$

$$4)\ a_{31}\ a_{12}\ a_{23};\quad 5)\ a_{21}\ a_{32}\ a_{13};\quad 6)\ a_{31}\ a_{22}\ a_{13}$$

from which any one possible is chosen.

As a result of studying the issue of constructing orthogonal cyclic permutations, it turned out that the construction of several orthogonal permutations (not necessarily cyclic), which are also used in solving transport-type problems with special restrictions, can be constructed using the method of solving the choice (assignment) problem. Let us describe Algorithm 2, which can be used to construct several orthogonal permutations of order *n*, in order to expand the possibilities of its application. If *n* permutations are built, then, in essence, this means that a Latin square [92], [143] of order *n* is built, which is widely used in scientific research, in economics, in agronomic research (to minimize the mutual influence of different experiments), in statistics, etc. Algorithm 2 will make it possible to use the elements of randomness in the construction of orthogonal permutations and, thereby, regulate the uniformity of the probability of each digit falling into each cell of the Latin square.

Algorithm 2. Construction of orthogonal permutations.

1) A square matrix *c* of order *n* is formed, all elements of which are *m*, $c = \{m\}$, *m* is a sufficiently large positive number, *m*>*n*. Matrix *LC* is a zero matrix of order *n*; upon completion of the calculations, all constructed orthogonal permutations will be presented in it;

2) In the cycle with respect to the variable *I*, *n* orthogonal permutations of order *n* are constructed; we put *C*=*c*. To construct each of the *n* permutations, a system of different representatives of the S_i, *i=1,2,…,n* sets is searched. At the first step of the cycle, with *I=1*, $S_i = (m, m, ..., m)$ for any *i*. At the next iterations, the S_i sets will turn into orthogonal permutations under construction.

3) According to the corresponding method, the minimum values of the rows are found, the matrix *C* is reduced by rows; then the minimum values of the columns are found and the matrix *C* is reduced by columns;

4) In the loop, "independent zeros" are determined (here, an element of chance can be used in the formation of a set of independent zeros);

5) If the number *k* of independent zeros is equal to *n*, then all representatives are found and we form the first permutation by them;

6) If not all independent zeros are found, then the matrix *C* transformation operation is performed, namely, the minimum set of lines that cross out all zeros of the matrix *C* is determined;

7) If there are not crossed out elements, the matrix C changes according to the rule:

$$\{C_{ij}\} \to \{C_{ij} - mnz\}, \quad (3)$$

where mnz is the minimum non-crossed out element of the matrix C.

Then the once crossed out elements of the matrix C from (3) are changed,

$$\{C_{ij}\} \to \{C_{ij} + mnz\}.$$

The rest, double-crossed-out elements of the matrix *C* remain unchanged.

After no more than *n* steps, we come to the end of the search for a complete system of various representatives, i.e. to the appointment system

$$\{p(i), a(i)\}, \quad i = 1, 2, ..., n;$$

8) We make adjustments to the matrices *c* and *LC*:

$$c(p(i), a(i)) = I, \; i = 1, 2, ..., n$$

$$LC(p(i), a(i)) = I, \; i = 1, 2, ..., n$$

where *I* is the cycle variable for the formation of permutations.

For example, if the dimension of the matrix is $n=7$, then after four iterations of the loop,

matrices c and LC could look like this:

$$c = \begin{pmatrix} M & M & M & m & m & m & m \\ M & M & m & m & m & m & M \\ M & m & M & m & m & M & m \\ m & m & M & M & m & m & M \\ m & M & m & M & M & m & m \\ m & m & m & M & M & M & m \\ m & m & m & m & M & M & M \end{pmatrix}$$

where M, m are arbitrary constant positive numbers, and M is greater than m, and m is greater, than n,

$$LC = \begin{pmatrix} 1 & 2 & 3 & 4 & 0 & 0 & 0 \\ 3 & 1 & 4 & 0 & 0 & 0 & 2 \\ 2 & 0 & 1 & 0 & 4 & 3 & 0 \\ 0 & 4 & 2 & 1 & 0 & 0 & 3 \\ 0 & 3 & 0 & 2 & 1 & 4 & 0 \\ 0 & 0 & 0 & 3 & 2 & 1 & 4 \\ 4 & 0 & 0 & 0 & 3 & 2 & 1 \end{pmatrix}$$

9) Let's move on to point 2).

As a result of this modification of the method for solving the choice problem, we obtain n orthogonal permutations, which together make up the Latin square [143] of order n, the LC marica at the end of the cycle might look like this:

$$LC = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 1 & 4 & 5 & 6 & 7 & 2 \\ 2 & 5 & 1 & 7 & 4 & 3 & 6 \\ 6 & 4 & 2 & 1 & 7 & 5 & 3 \\ 7 & 3 & 6 & 2 & 1 & 4 & 5 \\ 5 & 6 & 7 & 3 & 2 & 1 & 4 \\ 4 & 7 & 5 & 6 & 3 & 2 & 1 \end{pmatrix}$$

Algorithm 2 can be used in step 1 of algorithm 1 to construct two orthogonal cyclic permutations.

In conclusion of the section, we note that the research in this section is of interest from the point of view of developing new methods for solving problems of the transport type, for understanding the methods of operations research in solving economic problems.

4.3 Application of potentials method technology for solving the traveling salesman problem.

The research concerns the solution of the traveling salesman problem as a transport-type problem. The technology of the potentials method is used for the first time. Such a natural approach, due to the cyclicity and degeneracy of the solution to the traveling salesman problem, requires significant modification of the corresponding stages of the transport problem. An efficient algorithm for generating an initial cyclic solution has been developed; a criterion for the optimality of a solution is formulated; an algorithm for transition from the initial cyclic solution to another cyclic solution was constructed. To conduct mass computational experiments, an algorithm for constructing traveling salesman problems with a previously known optimal solution was used

One of the tasks of researching transport-type operations is the traveling salesman problem [137-146] It is natural to use the method of potentials to solve it. Consider the traveling salesman problem with cost matrix C of order n in the following interpretation. There are n points of departure for cargo $G_i, i = 1,...,n$ and shipment volumes for each point of consumption $a_i, i = 1,...,n$. Known need for cargo $b_j, j = 1,...,n$ for each of n destinations $H_j, j = 1,...,n$. The shipping cost for each option is The shipping cost for each option is $c_{ij}, i = 1,...,n; j = 1,...,n$

It is necessary to calculate a transportation plan with minimal transportation costs under the condition that all a_i and b_j are equal to one unit cargo, and the set of pairs of (i ,j) of points of departure i and destinations j of cargo constitute a cyclic substitution. One of the methods for solving the transport problem is the method of potentials [141,142,144-146], which includes the following steps: drawing up an initial feasible solution; calculation of potentials; checking the plan for optimality; search for the maximum value of the evaluation matrix; drawing up a cycle of reallocation of resources; determination of the minimum element in the redistribution contour and redistribution of resources along the contour; obtaining a

new feasible solution. The algorithm is carried out in several iterations until the optimal solution is found.

Let's consider algorithmization of the solution of the traveling salesman problem using the technology of the method of potentials.

When solving the traveling salesman problem by the method of potentials, it is necessary to take into account the differences between the traveling salesman problem and the usual transport problem at all stages of the algorithm. At the first stage of drawing up the initial plan, it is necessary to take into account that all volumes of sending $a_i = 1, i = 1,...,n$ and cargo needs at destinations $b_j = 1,\ j = 1,...,n$. All deliveries for each option, i.e. all variables x_{ij}, $i = 1,...,n$ and $j = 1,...,n$, that make up the feasible solution *x* of the traveling salesman problem, as for the transportation problem, must be positive.

One part of the decision variables *x* is equal to one, we will say that this is a set of ones. These variables make up the traveling salesman route. The set of ones is given by the cyclic substitution

$$(x, y) = \begin{pmatrix} x_1 x_2 ... x_n \\ y_1 y_2 ... y_n \end{pmatrix} \quad (1)$$

If $r = x_i, s = y_j,$, then $x_{rs} = 1.$ The number of such variables, based on the specifics of the problem, is *n*.

The other part of the variables of the feasible solution x, in the amount of n-1, are equal to zero, and these zeros are not ordinary, but fictitious, in order to prevent the degeneracy of the solution. Let's call this set the set of fictitious zeros and denote it by $(x\varepsilon, y\varepsilon) = \begin{pmatrix} x\varepsilon_1 x\varepsilon_2 ... x\varepsilon_{n-1} \\ y\varepsilon_1 y\varepsilon_2 ... y\varepsilon_{n-1} \end{pmatrix}$.

If $r = xe_i, s = ye_j,$, then $x_{rs} = \varepsilon,$ where ε is a sufficiently small positive number.

A cyclic permutation of $j_1 j_2 j_3 ... j_k j_{k+1} ... j_n$ corresponds to a traveling salesman route, i.e. a sequence of detour points, which corresponds to a type substitution

$$\begin{pmatrix} j_1 j_2 j_3 ... j_k j_{k+1} ... j_n \\ j_2 j_3 ... j_k j_{k+1} ... j_n j_1 \end{pmatrix}. \quad (2)$$

The substitution structure (2) shows one of the ways to generate n units of the initial cyclic feasible solution to the traveling salesman problem: generating

a random permutation $j_1 j_2 j_3 \ldots j_k j_{k+1} \ldots j_n$, i.e. row numbers x, and the formation of a permutation of $j_2 j_3 \ldots j_k j_{k+1} \ldots j_n j_1$, representing the numbers of the y columns, as in (1). So we determine the location of the units (*x, y*). The following algorithm 1 is used to generate a set of fictitious zeros (xe,ye) that provide the determination of all *2*n* potentials.

Algorithm 1.

1. We determine the matrix *x*, in which the numbers *M* are located diagonally - sufficiently large positive numbers, and in the positions determined by the set *(x, y)*, there are units; further, we supplement the matrix *x* to the matrix of an admissible solution;

2. Determine the first potential $u_1 = 0$ and the second potential $v_j = c_{1j}$, where *j* is the number of the column in which there is one in the first row of the matrix *x*, $x_{1j} = 1$;

3. Further in the cycle for *k, k=1,2,…,n-1*, each time, we determine two potentials u_{i_k} and v_{j_k}, we also enter one fictitious zero in the row with a random number i_k from the row numbers that do not yet contain a fictitious zero, and in the column j_{k-1}:

$$\begin{aligned} u_{i_k} &= c_{i_k j_{k-1}} - v_{j_{k-1}}, \\ x_{i_k j_{k-1}} &= \varepsilon, \\ v_{j_k} &= c_{i_k j_k} - u_{i_k}. \end{aligned} \qquad (3)$$

Remark 1. The indices of the last and first potentials determine the position of the additional n-th fictitious zero, which can be used as previously defined fictitious zeros.

Remark 2. Algorithm 1, as a rule, generates a cyclic substitution from positions of n fictitious zeros. If the first attempt does not result in a cyclic substitution, you can make the next attempt until one is obtained. Another variant of Algorithm 1 for obtaining a cyclic substitution is to enumerate possible options for calculating some 2-3 last potentials.

The generation of fictitious zeros essentially determines the quality of the initial solution, i.e. remoteness of the initial solution from the optimal one.

Theorem 1. Let *(xn,yn)* be any initial cyclic route of the traveling salesman, and *(xo,yo)* be the optimal cyclic route. There is a set $(x\varepsilon, y\varepsilon)$ of n-1 fictitious zeros such that, given an admissible solution *x*, consisting of *n*

elements *(xn,yn)* and *n-1* fictitious zeros $(x\varepsilon, y\varepsilon)$, it is possible to construct a transition cycle from the initial *(xn, yn)* to the optimal route *(xo, yo)*.

Proof. Suppose we have generated an initial cyclic route

$$(xn, yn) = \begin{pmatrix} xn_1 xn_2 ... xn_n \\ yn_1 yn_2 ... yn_n \end{pmatrix}, \tag{4}$$

optimal route

$$(xo, yo) = \begin{pmatrix} xo_1 xo_2 ... xo_n \\ yo_1 yo_2 ... yo_n \end{pmatrix}. \tag{5}$$

in which possibly s positions of units (4) coincide with *s* positions of units of the optimal route (5).

Since the sequence of writing pairs $\begin{pmatrix} xn_i \\ yn_i \end{pmatrix}$ and $\begin{pmatrix} xo_j \\ yo_j \end{pmatrix}$ in (4) and (5), respectively, is not significant, we can assume, for s not equal to zero, that the sets $\begin{pmatrix} xn_1 xn_2 ... xn_s \\ yn_1 yn_2 ... yn_s \end{pmatrix}$ and $\begin{pmatrix} xo_1 xo_2 ... xo_s \\ yo_1 yo_2 ... yo_s \end{pmatrix}$ match. For the time being, we exclude these s units from consideration, and for the remaining

$$\begin{pmatrix} xn_{s+1} xn_{s+2} ... xn_n \\ yn_{s+1} yn_{s+2} ... yn_n \end{pmatrix} \text{ and } \begin{pmatrix} xo_{s+1} xo_{s+2} ... xo_n \\ yo_{s+1} yo_{s+2} ... yo_n \end{pmatrix}$$

let's make a chain from the first unit of the initial route to the unit along the line of the optimal route, and then along the column from the unit of the optimal route to the unit of the initial route.

Having made such a transition *n-s* times from the unit of the initial route to the unit of the optimal route, we close the loop. Again, referring to the arbitrariness of the sequence of writing the positions of units in (4) and (5), we can write the resulting cycle as follows:

$$\begin{pmatrix} xn_{s+1} \\ yn_{s+1} \end{pmatrix} \begin{pmatrix} xo_{s+1} \\ yo_{s+1} \end{pmatrix} \begin{pmatrix} xn_{s+2} \\ yn_{s+2} \end{pmatrix} \begin{pmatrix} xo_{s+2} \\ yo_{s+2} \end{pmatrix} ... \begin{pmatrix} xn_n \\ yn_n \end{pmatrix} \begin{pmatrix} xo_n \\ yo_n \end{pmatrix} \tag{6}$$

Now we will single out one of the even positions, for example, $\begin{pmatrix} xo_j \\ yo_j \end{pmatrix}$, as a free variable, and in the remaining even positions (they are in the amount of *n-s-1*) from (6) we will put fictitious zeros in the matrix *x*. At the end, we introduce s more fictitious zeros, arbitrarily chosen, but complementing all even positions (6) to a cyclic set. $(x\varepsilon, y\varepsilon)$. At the end, we introduce *s* fictitious zeros,

arbitrarily chosen, but complementing all even positions (6) to the cyclic set $(x\varepsilon, y\varepsilon)$.

The admissible solution of the transport problem constructed in this way, starting from the free variable $\begin{pmatrix} xo_j \\ yo_j \end{pmatrix}$ and going through the cycle (6), transforms the initial route into the optimal route, consisting of *s* units in coinciding positions and *n-s* units in even positions (6).

Let us demonstrate Theorem 1 by an example. Assume that the initial route, i.e. set of units (sequence of bypass points) $(xn, yn) = \begin{pmatrix} 83675124 \\ 36751248 \end{pmatrix}$ and the second optimal route $(xo, yo) = \begin{pmatrix} 63785124 \\ 37851246 \end{pmatrix}$. The common positions of the routes are (1,2), (2,4) and (5,1), *s=3*.

For the remaining positions of units in both routes, we build a cycle. Let's start, for example, from position (3,6) and further along the row, and then along the column, etc., we get a cycle

(3,6), (3,7), (6,7), (6,3), (8,3), (8,5), (7,5), (7,8), (4,8), (4,6).

Let us replace the transition (3.6) with (3.7), (6.7) with (6.3), (8.3) with (8.5), (7.5) with (7.8) and (4.8) to (4.6). As a result of such replacements, taking into account the coinciding *s* positions, we get the transition from the cyclic route *(xn,yn)* to the optimal cyclic route *(xo, yo).*

Theorem 1 shows the possibility of solving the traveling salesman problem, as a transport type problem, method of potentials. To further explain the corresponding Algorithm 2, we will continue to follow the steps of solving the usual transportation problem in relation to solving the traveling salesman problem.

At the second stage, according to the admissible solution *x* of the traveling salesman problem as a transport problem, i.e. for *x*, compiled by Algorithm 1 from a cyclic route *(xn,yn)* and a set of fictitiouszeros $(x\varepsilon, y\varepsilon)$, allowing you to calculate all *2 * n* potentials u_i and v_j, the matrix of estimates is determined

$$m = \{m_{ij}\}, \tag{7}$$

where $m_{ij} = u_i + v_j - c_{ij}, \quad i, j = \overline{1,n}$

The so-defined evaluation matrix *m*, we will say that corresponds to the route *(xn,yn)* and set of fictitious zeros $(x\varepsilon, y\varepsilon)$.

At the stage of estimating the optimality of the solution to the traveling salesman problem, the following theorem is used.

Theorem 2. Let *(xn,yn)* be a cyclic route and $(x\varepsilon, y\varepsilon)$ be the set of fictitious zeros used to calculate the evaluation matrix *m*. In order for *(xn,yn)* to be an optimal solution to the traveling salesman problem, it is necessary that the sum *S* of estimates on this route be nonpositive, i.e.

$$S = \sum_{i=1}^{n} m_{xn_i yn_i} \le 0. \tag{8}$$

Proof. Let us assume that (*xn,yn*) is the optimal solution and at the same time S>0.

Let us calculate the difference between the value of the objective function on the route (*xn,yn*) and the sum of the estimates on this route,

$$\sum_{i=1}^{n} c_{xn_i yn_i} - S =$$

$$= \sum_{i=1}^{n} u_{xn_i} + \sum_{i=1}^{n} v_{yn_i} - \sum_{i=1}^{n} m_{xn_i yn_i} =$$

$$= \sum_{i=1}^{n} u_{xn_i} + \sum_{i=1}^{n} v_{yn_i} - (\sum_{i=1}^{n} u_{xn_i} + \sum_{i=1}^{n} v_{yn_i} - \sum_{i=1}^{n} c_{xn_i yn_i}) = \sum_{i=1}^{n} c_{xn_i yn_i}.$$

We have obtained a contradiction, which means that the sum of estimates S must not be positive.

Theorem 3. Condition (8) is sufficient for the optimality of *(xn,yn)* provided that $(x\varepsilon, y\varepsilon)$ is any set of fictitious zeros.

Proof. According to the admissible solution *x*, formed from two sets *(xn,yn)* and $(x\varepsilon, y\varepsilon)$, for the corresponding transport problem, we calculate the matrix of estimates *m*. Assume that the optimal route is not *(xn,yn)*, but some other route (xd, yd). with $S \le 0$ with the same set of fictitious zeros $(x\varepsilon, y\varepsilon)$.

Then the following equalities hold

$$\sum_{i=1}^{n} c_{xn_i yn_i} - S =$$

$$= \sum_{i=1}^{n} u_{xn_i} + \sum_{i=1}^{n} v_{yn_i} - \left(\sum_{i=1}^{n} u_{xd_i} + \sum_{i=1}^{n} v_{yd_i} - \sum_{i=1}^{n} c_{xd_i yd_i} \right) =$$

$$= \sum_{i=1}^{n} c_{xd_i yd_i}$$

Hence from the optimality of (xd, yd) and from the fact that $S \le 0$, follows the optimality of the route *(xn,yn).*

Thus, the stage of searching for the maximum value of the assessment matrix when solving the transport problem is replaced by searching for the maximum value of the sum of the assessment matrix on a given cyclic route (xn,yn) when solving the traveling salesman problem.

Other stages of solving the transport problem by the method of potentials, namely, drawing up a cycle of resource redistribution, determining the minimum element in the redistribution contour and redistributing resources along the contour, obtaining a new feasible solution, are replaced, when solving the traveling salesman problem, with the route search stage *(xn,yn)* with the maximum value of the sum of estimates on it. If, as a result of the analysis of the estimates of the matrix m, there is an (xd, yd) route with a total estimate S>0, then such a route is better than the initial one.

As for solving any transport type problem, it is necessary to use some way to generate an initial solution. Considering the specifics of the traveling salesman problem, the following efficient algorithm 2 is proposed.

Algorithm 2.

1.Ordering the matrix with in ascending order. We get data set *D* in the form

$$c_{i_r i_s}, i_r, i_s,\ r \ne s,\ r, s = \overline{1,n}$$

2. Construction of a cyclic permutation of $(i_1, i_2, ..., i_n)$ for each element of the $c_{i_r i_s}$ from *D*.

The first transition (i_1, i_2) is determined from the equality $c_{i_1 i_2} = c_{i_r i_s}$ whence it follows $i_1 = i_r$ and $i_2 = i_s$.

3. If the first *t-1* pairs of $(i_1, i_2), ..., (i_{t-1}, i_t)$ are defined, then the next pair of (i_t, i_{t+1}) is determined from the condition: the $c_{i_t i_{t+1}}$ element is the first element (in the original ordered set *D*) for which the i_{t+1} is not equal to any of the $i_1, i_2, ..., i_t$ numbers.

4. From all *n(n-1)* cyclic permutations constructed in steps 2 and 3, a permutation with a minimum value of the objective function is selected, which is taken as the initial solution.

Based on the foregoing, Algorithm 3 is constructed to find a solution to the traveling salesman problem.

Algorithm 3.

1.Input of initial data: cost matrices *c* and its dimension *n*.

2.Generation of the initial route in the form of a cyclic substitution (*xn,yn*), for example, using algorithm 2.. Simultaneously, we form a matrix x of size n with elements equal to zero, except for $x(xn_i, yn_i) = 1, i = \overline{1,n}.$

3. We determine *f*, the cost of the route, i.e. the value of the objective function on the substitution *(xn,yn)*, $f = \sum_{i=1}^{n} c_{xn_i yn_i}$ and remember the values of *f and (xn,yn), fop=f, xop=xn, yop=yn.*

4. We perform a predetermined number of iterations of finding a better route than *(xop.yop)*.

4.1. Given (*xn,yn*) and *c*, we generate a set of fictitious zeros $(x\varepsilon, y\varepsilon)$ so that by the matrix x, complemented by the values $x(x\varepsilon_i, y\varepsilon_i) = \varepsilon, i = \overline{1,n}$ it was possible to calculate all *2n* potentials $u_i, v_j, i, j = \overline{1,n}$, ε is a fictitious zero, a sufficiently small positive number.

4.2. For the values of *n, c, u, v* now determined, we calculate the matrix of estimates

$$m = \{m_{ij}\},\ i, j = \overline{1,n},$$

$$m_{ij} = \begin{cases} u_i + v_j - c_{ij}, & i \neq j \\ -M, & i = j \end{cases},$$

where *M* is a sufficiently large positive number.

In the evaluation matrix *m*, all $m(xn_i, yn_i), i = \overline{1,n}$ and $m(x\varepsilon_j, y\varepsilon_j), j = \overline{1,n-1}$ elements are equal to zero.

The remaining elements are positive or negative numbers. If there are no positive elements, then the route *(xn,yn)* is optimal, and the calculations end. In the presence of both positive and negative elements, as in solving the transport problem, a search is made for the best route.

4.3. Search for a route with a positive sum of ratings.

4.3.1. After sorting all the values of the assessment matrix m, except for the diagonal ones, in descending order, we get the following data:

$$y_k, iy_k, jy_k, k=1, 2, \ldots, n(n-1), \quad (9)$$

where $y_k = m(iy(k), jy(k))$. Let's put *k=0*.

4.3.2 For the *k+1* iteration, a cyclic substitution

$$(xn, yn) = \begin{pmatrix} i_1 i_2 ... i_{s-1} i_s ... i_n \\ i_2 i_3 ... i_s i_{s+1} ... i_1 \end{pmatrix} \quad (10)$$

is constructed, where the first (i_1, i_2) pair coincides with the (iy_r, jy_r) pair from (9).

Let's assume that the first *s-1* pairs $(i_1, i_2), (i_2, i_3), ..., (i_{s-1}, i_s)$ in (10) are defined. Next couple (i_s, i_{s+1}), at www, coincides with the pair (iy_r, jy_r) from (9) under the conditions that

a) iy_r does not match any of $i_1, i_2, ..., i_{s-1}$;

b) jy_r does not match any of $i_1, i_2, ..., i_s$.

For $s = n$ pair (i_s, i_{s+1}) coincides with pair (i_n, i_1) with i_n and i_1 already defined.

4.3.3. For the constructed route *(xn,yn),* the score *S* is calculated as the sum of the elements of the evaluation matrix *m,*.

$$S = \sum_{i=1}^{n} m(xn_i, yn_i) .$$

If $S > 0$, then go to item 3, otherwise go to item 4.3.2.

5. If at step 4.3.2 it was not possible to find the best route, with an estimate of S>0 , then go to step 2, to perform the next iteration.

6. Upon completion of the specified number of iterations, at the output we get the route with the minimum cost, obtained as a result of selection from the routes generated by the algorithm.

At the stages of constructing the initial cyclic solution and moving from the initial to the new cyclic solution, the following algorithm is applied, along with Algorithm 3, in relation to the original matrix c and the evaluation matrix m.

Algorithm 4. Let a cyclic permutation $I = (i_1, i_2, i_3, \ldots, i_n)$ be given. Let's form pairs

$$(i_1, i_2), (i_2, i_3), (i_3, i_4), (i_4, i_5), \ldots, (i_{n-1}, i_n), (i_n, i_{n+1}), \quad i_{n+1} = i_1, \quad (11)$$

which make up the route, and take from them any three pairs

$$(i_{k_1}, i_{s_1}), (i_{k_2}, i_{s_2}), (i_{k_3}, i_{s_3}) \quad (12)$$

Let's replace pairs (12) with pairs

$$\left(i_{k_1}, i_{s_2}\right), \left(i_{k_2}, i_{s_3}\right), \left(i_{k_3}, i_{s_1}\right) \tag{13}$$

substitute them into (11) and get a new route

$$\left(j_1, j_2\right), \left(j_2, j_3\right), \left(j_3, j_4\right), \left(j_4, j_5\right), \ldots, \left(j_{n-1}, j_n\right), \left(j_n, j_{n+1}\right), \quad j_{n+1} = j_1, \tag{14}$$

that corresponds to a cyclic permutation corresponds to a cyclic permutation $J = \left(j_1, j_2, j_3, \ldots, j_n\right)$

Comparing values $f_1 = \sum_{k=1}^{n} c(i_k, i_{k+1})$ and $f_2 = \sum_{k=1}^{n} c(j_k, j_{k+1})$ and choose the best of solutions *I* and *J*, minimum for matrix c. In the case of applying algorithm to matrix m, we select the maximum value.

Calculations according to Algorithm 3 were performed with traveling salesman problems, which were generated with a pre-selected optimal solution. The following question was considered. Assume that for the traveling salesman problem A with cost matrix *c*, the optimal solution is *(xa,ya)*. What needs to be changed in matrix c to get the traveling salesman problem B with cost matrix *d* and optimal solution *(xb,yb)*? We formulate the answer in the form of Theorem 4 and give its constructive proof.

Theorem 4. Let the *c* matrix of dimension *n* consist of non-negative integers. On the basis of matrix *c*, one can construct a traveling salesman problem with matrix *d* with a pre-selected optimal solution *(xb,yb)*.

Proof. Given the matrix c and the solution *(xb,yb)*, we determine (for example, by algorithm 1) a set of *n-1* fictitious zeros . Let us compose a matrix *x*, in which some of the elements are equal to one, $x(xb, yb) = 1, \quad i = \overline{1,n}$, another part of the elements from fictitious zeros, another part of the elements from fictitious zeros, $x(x\varepsilon_j, y\varepsilon_j) = \varepsilon, j = \overline{1,n-1}$, ε - a small enough positive number. We set the diagonal elements of the matrix *x* to be equal to the *M* - a large enough positive number. $x(i,i) = M, i = \overline{1,n}.$ The remaining elements of the matrix *x* are equal to zero. From the matrix *c* and *x*, we calculate the potentials $u_i, i = \overline{1,n}$, and $v_j, j = \overline{1,n}$, and the corresponding evaluation matrix m,

$$m = m(i,j) = \begin{cases} u_i + v_j - c_{ij}, i \neq j, \\ -M, \ i = j \end{cases}, \ i, j = \overline{1,n}.$$

The matrix *d* of the traveling salesman problem with the optimal solution to (xb, yb) is determined by the formula

$$d = d(i,j) = \begin{cases} u_i + v_j + \alpha_{ij}, \text{ at } i \neq j, x(i,j) = 0, m(i,j) > 0 \\ c_{ij}, \text{ otherwise} \end{cases}$$

where α_{ij} are non-negative integer random numbers.

The optimality of the (xb, yb) solution follows from the fact that, according to the optimality criterion of Theorem 2, it is impossible to obtain a positive total estimate for any cyclic solution.

Let us give one example and the results of some experiments with Algorithm 3.

3 Computational experiments.

Example. Let the cost matrix, dimension n=8, for the traveling salesman problem be

c=[M 76 43 38 51 42 19 80; 42 M 49 26 78 52 39 87;
48 28 M 36 53 44 68 61; 72 31 29 M 42 49 50 38;
30 52 38 47 M 64 75 82; 66 51 83 51 22 M 37 71;
77 62 93 54 69 38 M 26; 42 58 66 76 41 52 83 M],

Starting route $(xn, yn) = \begin{pmatrix} 65174832 \\ 51748326 \end{pmatrix}$ has a value of $f = \sum_{i=1}^{n} c(xn_i, yn_i) = 309$. We generate n-1 fictitious zeros, for example, by algorithm 1. $(x\varepsilon, y\varepsilon) = \begin{pmatrix} 2345678 \\ 3158467 \end{pmatrix}$, then we calculate the potentials and the evaluation matrix m,

m=[-M -160 -41 -17 -59 -37 0 -92; -59 -M 0 42 -39 0 27 -52;
0 0 -M 97 51 73 63 39; -86 -65 23 -M 0 6 19 0;
0 -42 58 68 -M 35 38 0; -100 -105 -51 0 0 -M 12 -53;
-108 -113 -58 0 -44 0 -M -5; -42 -78 0 9 15 17 0 -M];

At the first iteration, from the analysis of the matrix m, we obtain a positive sum of estimates $S = \sum_{i=1}^{n} m(xn_i, yn_i) = 2$ on the $(xo, yo) = \begin{pmatrix} 17653248 \\ 76532481 \end{pmatrix}$ route. In this case, the new value of the objective function is equal to the difference between the previous value of f and the sum of the estimates S, $f - S = 307$. A new iteration is carried out with the best solution. If S<=0, then we start a new iteration with the previous solution. So, after the 13th iteration, we arrive at the optimal solution of the traveling salesman problem (xo, yo) with aaa value f=251. This means that by calculating the evaluation matrix m

for the optimal solution *(xo,yo)* and any set of fictitious zeros , we will not get any solution of the traveling salesman problem *(xn,yn)* with a positive total estimate $S = \sum_{i=1}^{n} m(xn_i, yn_i)$, despite the presence of positive elements in the assessment matrix m, as in solving the transport problem.

In the experiments, the numbers of α_{ij} were taken from the range from zero to 50. For the optimal solutions obtained for problems with $n \le 16$ in no more than 100 iterations, the $\Delta f = \left(f - f_{opt} \right) / f_{opt}$ score did not exceed fifty persent. With an increase in the number of iterations, the number of solved problems for which $\Delta f = 0$ increases.

The results of computational experiments show that the use of potential method technology to solve the traveling salesman problem, as a special transport problem, is a new promising direction in searching for a high-quality solution.

4.4 On one optimality criterion for solving the traveling salesman problem

The study relates to the problem of a traveling salesman, which consists in finding the most profitable route passing through each of the specified cities once and then returning to the original city. The purpose of a bypass is to minimize the function associated with transportation costs, distance traveled, or time spent.

According to its mathematical formulation, the traveling salesman problem is a transport type problem. It occurs in many situations that are not very similar to transport ones: in computer science to optimize data paths between different nodes, to create cost-effective manufacturing, to plan drone missions, to develop better software systems useful for technologies such as self-driving cars, and helping companies save money. The need to solve the traveling salesman problem arises when developing software for processing large amounts of information.

In general, methods for solving the traveling salesman problem can be considered approximate, not always providing an optimal solution in an acceptable time, with the exception of the exhaustive search method for small problems.

The purpose of the research is to develop a computational algorithm for solving the traveling salesman problem, based on the use of a three-element

replacement operation and technology for solving transport problems using the potentials method.

Let us present the results. Let c be the matrix of transport costs, $x_{a_1b_1}, x_{a_2b_2}, ..., x_{a_nb_n}$ arbitrary admissible cyclic solution to the traveling salesman problem. We form combinations of three non-zero elements from this solution. If for any combination $x_{i_1 j_1} = 1$, $x_{i_2 j_2} = 1$ and $x_{i_3 j_3} = 1$ the following relation holds

$$c_{i_1 j_1} + c_{i_2 j_2} + c_{i_3 j_3} \le min\ (c_{i_2 j_1} + c_{i_3 j_2} + c_{i_1 j_3},\ c_{i_3 j_1} + c_{i_1 j_2} + c_{i_2 j_3}),$$

then the solution to the problem has been found. If

$$c_{i_1 j_1} + c_{i_2 j_2} + c_{i_3 j_3} > min\ (c_{i_2 j_1} + c_{i_3 j_2} + c_{i_1 j_3},\ c_{i_3 j_1} + c_{i_1 j_2} + c_{i_2 j_3}),$$

then a three-element replacement operation is performed $x_{i_2 j_1} = 1$, $x_{i_3 j_2} = 1$, $x_{i_1 j_3} = 1$ or $x_{i_3 j_1} = 1$, $x_{i_1 j_2} = 1$, $x_{i_2 j_3} = 1$ depending on which of the two terms is smaller and, in any case,

$$x_{i_1 j_1} = 0,\ x_{i_2 j_2} = 0,\ x_{i_3 j_3} = 0.$$

This three-element replacement operation is used to obtain a new feasible cyclic solution with a lower cost value, and the solution is again tested for the final one.

The criterion for the optimality of a solution is that for any cyclic solution it is impossible to carry out a three-element replacement operation. Based on this optimality criterion, it is possible to construct schemes for solving the traveling salesman problem within the framework of the potentials method technology for solving the transport problem. For example, for a positive element of the evaluation matrix corresponding to the zero element of the matrix x, a cycle is constructed

$$(i_1, j_1), (i_2, j_2), (i_3, j_3), (i_4, j_4), ..., (i_{k-1}, j_{k-1}), (i_k, j_k),$$

where k is even and $k \le 2n$. Next, the unit elements of the matrix x are selected, standing at the even positions (i_2, j_2), (i_4, j_4), ..., (i_k, j_k) of this cycle. In this way, a cyclic (for $k=2n$) or partially cyclic (for $k<2n$) solution is formed

$$x_{i_2 j_2} = 1, x_{i_4 j_4} = 1, ..., x_{i_k j_k} = 1,$$

for which the fulfillment of the optimality criterion is checked.

Testing of algorithms built on the basis of the optimality criterion and massive computational experiments were carried out on automatically generated traveling salesman problems with a previously known optimal solution [141-146].

The conducted computational experiments based on the optimality criterion showed that the developed schemes for solving traveling salesman problems can be classified as optimal in terms of calculations due to the efficiency of the three-element replacement operation and the choice of different initial cyclic solutions at each iteration of the modified distribution method. Such algorithms are recommended for use for the accelerated and acceptable solution of traveling salesman problems in conditions of processing a sufficiently large amount of information.

4.5 On one implementation of a natural approach to solving the traveling salesman problem

The work constructs a new computational scheme for solving the traveling salesman problem as a transport-type problem. The technology used is based on a modified distribution method. The concept of a three-element replacement operation is introduced, which is used to formulate the necessary condition for the optimality of a solution. The search for the optimal solution begins with a random cyclic route. The solution is then improved by applying a three-dimensional replacement operation. There is a transition from the current cyclic solution to a new cyclic solution with a better value of its cost. The search for an optimal solution ends when the three-dimensional replacement operation fails after a given number of iterations. Computational experiments have shown the possibility of obtaining an acceptable solution for problems of a sufficiently large size. This work represents a further development of the natural approach to solving the traveling salesman problem.

The traveling salesman problem has been the object of numerous studies, which resulted in works concerning the development of heuristic methods, the branch and bound method, the genetic method, the ant colony method, the use of neural networks, etc. [53,56,57,59-61]. The versatility of approaches to solving the traveling salesman problem is relevant for creating a complete theory of it.

The here presents one of the implementations of the idea of using a modified distribution method (potentials method) to solve the traveling salesman problem. This approach is natural, since the traveling salesman problem is a transport-type problem. Direct application of the modified distribution method to solve the traveling salesman problem is impossible, since the corresponding transport problem is degenerate and, in addition, due to the requirement that the solution be cyclic. Eliminating degeneracy and taking into

account the cyclicity of an admissible solution is a prerequisite for the development of algorithms that use computer technology for solving the traveling salesman problem using the specified method.

4.6. Modification of the potentials method in solving the traveling salesman problem

The use of the potentials method as a type of method of sequential improvement of the plan in relation to the transport problem is possible by modifying the corresponding stages of solving the transport problem, including the generation of the initial feasible solution, calculation of the evaluation matrix, assessment of the optimality of the solution, and transition to a new feasible solution.

The necessary changes, in view of the requirement of cyclicity and non-degeneracy of the solution to the traveling salesman problem as a transport problem, are set out further in Algorithms 1–3 (sometimes using MATLAB language tools [62]), are as follows.

At the initial stage, a cyclic solution (route) is generated. The degeneracy of the solution is eliminated by a special method of introducing n–1 fictitious zeros ε (ε is a fairly small value of the transported goods).

The potentials are calculated simultaneously with the introduction of fictitious zeros. The introduction of a fictitious zero in position (*i*, *j*) is associated with the simultaneous calculation of the potential v_j for column j and the potential u_i for row *i*, which ensures the non-degeneracy of the initial cyclic solution. Then the evaluation matrix, $m_{ij} = u_i + v_j - c_{ij}$, is calculated.

The use of the matrix *m* to assess the optimality of the current solution, in the case of solving the traveling salesman problem, is carried out not according to the criterion of non-positive elements of m, but in a different way due to the possibility of breaking the duality.

At each iteration of the algorithm, a transition occurs from a cyclic solution (route) (*xv*, *yv*), using the replacement of its three constituent elements (points), to a new cyclic solution (*xv*2, *yv*2).

Definition. The operation of transition from one cyclic solution, with the replacement of three elements in it, to another cyclic solution, will be called a three-element replacement operation.

The use of a three-element replacement operation is intended to move from one cyclic solution to another cyclic solution, as in the fbsol function:

[*fopt*, *xv*2, *yv*2, *cucr*]=***fbsol*** (*n*, *c*, *x*, *xv*, *yv*, *muv*, *Inf*).

With the completion of Algorithm 1, the best cyclic solution is determined, found in a certain number of iterations with different initial cyclic solutions.

4.6.1 Description of algorithms and functions

The description of the main algorithm 1 includes a function for generating the initial route and the initial cyclic solution, a function for calculating the evaluation matrix, and a function for transitioning from the initial cyclic solution to a new cyclic solution using a three-element replacement operation.

Algorithm 1 and its constituent functions

a) An initial random cyclic route is formed (*xv*, *yv*),

$$xv = \textbf{\textit{randperm}}(n);\ yv = [xv(2{:}n),\ xv(1)],$$

auxiliary matrix *X*,

$$X(1{:}n,1{:}n) = 0;\ for\ i = 1{:}n\ \ X(xv(i), yv(i)) = 1;\ X(i,i) = Inf;$$

and solution

$$xvop = [xv];\ yvop = [yv]\ with\ fop = f,$$

where $f = \sum_{i=1}^{n} c(xv(i), yv(i))$;

b) Use the ***fmdr*** function,

$$[u, v, x, j1, puv] = \textbf{\textit{fmdr}}\ (n, X, c),$$

with parameters *n*, *X* and *c* for simultaneous calculation of an admissible solution *x* and the corresponding potentials u_i and v_j, *i* = 1,2,..., *n*, *j* = 1,2,..., *n*. The fmdr function ensures that the solution *x* is ondegenerate and that all 2*n* potentials can be determined. We calculate the matrix of estimates *muv* and *tm* – the number of positive elements *tm* in it:

$$[muv, tm] = \textbf{\textit{MatrPot}}\ (n, x, c, u, v).$$

If the evaluation matrix does not contain positive elements, *tm* = 0, then solution *x* is optimal. Such a case may occur, for example, when the optimal solution to the assignment problem with matrix *c* is a cyclic solution;

c) Otherwise, which happens more often when duality breaks, when *tm* > 0, we continue searching for the best solution *x* using the function ***fbsol***,

$$[fopt, xv2, yv2, cucr] = \textbf{\textit{fbsol}}\ (n, c, x, xv, yv, muv, Inf);$$

The output parameters mean the best cyclic solution (*xv*2, *yv*2) obtained at the current iteration, with the value of the objective function fopt after cucr improvements of the cyclic solution.

If *fopt* < *fop*, then we form the best solution for all iterations

$$xvop = [xv2];\ yvop = [yv2]$$

with $fop = fopt$;

d) If $cucr = 0$, calculations end, otherwise start the next iteration, in which as the initial solution we take the best solution found

$$xv = [xvop],\ yv = [yvop]$$

and go to step b).

Let us describe the content of the functions used.

Algorithm 2 for constructing a non-degenerate cyclic solution.

The formation of the initial cyclic solution x is carried out by the function fmdr,

$$[u, v, x] = fmdr(n, X, c).$$

a) At the first iteration, having formed a random cyclic substitution (xv,yv) and its matrix representation X, we determine the first zero potential $u(1)$ and the k-th potential $v(k) = c(1,k)$, where $X(1, k) = 1$;

b) At the next iterations, in one of the rows with a random number r, $r = 2,...,n$, for which the potential $u(r)$ has not yet been determined, we introduce one fictitious zero in column k and row r, $X(r,k) = \varepsilon$, and calculate the potential $u(r) = c(r,k) – v(k)$. We find in row r the number s of the element $X(r,s)$ equal to one, and set $v(s) = c(r,s) – u(r)$;

c) Taking as k the number of the last column s with the already determined potential $v(s)$, we go to step b).

Algorithm 3. Transition from a cyclic solution (xv, yv) to a new cyclic solution.

Let the admissible cyclic solution x and the evaluation matrix muv be known.

a) For each zero element $x(i, j)$, which corresponds to a positive value of the evaluation matrix muv, we find in row i and column jc a matrix element x equal to one, $x\ (i, jc) = 1$. Also, using column j we find the row ic with the identity element, $x(ic, j) = 1$.We put $a1 = i$, $b1 = jc$, $a2 = ic$, $b2 = j$;

b) In the loop, for each $i1 = 1 : n$, with $x(xv(i1), yv(i1)) = 1$ and $(xv(i1), yv(i1))$ not matching $(a1,b1)$ and $(a2,b2)$ determine

$$a3 = xv(i1),\ b3 = yv(i1);$$

$$ci(1) = a1;\ cj(1) = b1;\ ci(4) = a2;\ cj(4) = b1;\ ci(7) = a3;\ cj(7) = b1;$$
$$ci(2) = a2;\ cj(2) = b2;\ ci(5) = a3;\ cj(5) = b2;\ ci(8) = a1;\ cj(8) = b2;$$
$$ci(3) = a3;\ cj(3) = b3;\ ci(6) = a1;\ cj(6) = b3;\ ci(9) = a2;\ cj(9) = b3;$$

$$sc0 = c(a1,b1) + c(a2,b2) + c(a3,b3);$$
$$sc1 = c(a2,b1) + c(a3,b2) + c(a1,b3);$$
$$sc2 = c(a3,b1) + c(a1,b2) + c(a2,b3);$$

$$[msc,isc]=min([sc1\ sc2]);$$

c) If $sc0 > msc$ and $msc < Inf$, we determine the solution $(xv1,yv1)$,

$$xv1 = [xv];\ yv1 = [yv],$$

and then we adjust it depending on at what isc the minimum value is reached; when $isc=1$

$$xv1(j1) = ci(i1+3),\ yv1(j1) = cj(i1+3),$$

at $isc = 2$

$$xv1(j1) = ci(i1+6),\ yv1(j1) = cj(i1+6).$$

From all new solutions $(xv1, yv1)$, the solution with the minimum cost is selected, which is considered the result of Algorithm 3,

$$fopt = f;\ \ xv2 = [xv1];\ \ yv2 = [yv1].$$

4.6.2 Rationale for algorithms

Calculations according to Algorithm 1 are made with matrix X, which represents an admissible non-degenerate solution to the traveling salesman problem as a transport-type problem with unit volumes of consumption of a homogeneous product for n consumers and unit volumes of product production at n production points.

The matrix X is formed from n 1s and n–1 dummy zeros, with one 1 and one dummy zero in each column and each row, except for one column and one row, which contain only one 1.

The cyclic route is interpreted as a cyclic substitution $(x1, y1)$ or as a matrix X containing identity elements at positions $(x1, y1)$ and dummy elements at positions $(x\varepsilon, y\varepsilon)$

$$X(x1(i), y1(i)) = 1,\ \ i = 1, 2, \ldots, n,\ \ X(x\varepsilon(j), y\varepsilon(j)) = \varepsilon,\ \ j = 1, 2, \ldots, n-1.$$

Three-element replacement operation means transforming one cyclic route to another cyclic route as follows. Let the zero element $X(i0, j0)$, $i0 \neq j0$, of the matrix X correspond to a positive value of the evaluation matrix muv. Let's take three unit elements of matrix X, for example from rows $i1$, $i2$ and $i3$ and columns $j1$, $j2$ and $j3$

$$X(i1, j1) = X(i2, j2) = X(i3, j3) = 1.$$

The three-element replacement operation consists of zeroing the values of X in $(i1, j1)$, $(i2, j2)$ and $(i3, j3)$,

$$X(i1, j1) = X(i2, j2) = X(i3, j3) = 0$$

and entering three elements in new positions:

$$X(i2, j1) = X(i3, j2) = X(i1, j3) = 1 \text{ when } s0 < s1 \leq s2 \qquad (1)$$

or

$$X(i3, j1) = X(i1, j2) = X(i2, j3) = 1 \text{ when } s0 < s2 < s1, \quad (2)$$

where

$$s0 = c(i1, j1) + c(i2, j2) + c(i3, j3),$$
$$s1 = c(i2, j1) + c(i3, j2) + c(i1, j3), \quad (3)$$
$$s2 = c(i3, j1) + c(i1, j2) + c(i1, j3).$$

Let

$$(i1, j1), (i2, j2), (i3, j3), \ldots, (in–1, jn–1), (in, jn) \quad 4)$$

optimal route for the traveling salesman problem.

Theorem. For any three pairs

$$(ip, jq), (ir, js), (iu, jv) \quad (5)$$

from the optimal route (4), inequality

$$c(ip, jq) + c(ir, js) + c(iu, jv) \leq min\ (c(ir, jq) + c(iu, js) + c(ip,jv), \quad (6)$$
$$c(iu, jq) + c(ip, js) + c(ir, jv))$$

is true.

Proof. The three-element replacement operation converts the current cyclic route into a new cyclic route with a lower cost. If route (4) is optimal, then inequality (6) must be true. Otherwise, if

$$c(ip, jq) + c(ir, js) + c(iu, jv) > c(ir, jq) + c(iu, js) + c(ip, jv),$$

then in route (4) all three elements of matrix X in positions (5) can be replaced by elements X in positions (ir, jq), (iu, js), (ip, jv) and obtain a new cyclic optimal a route different from the original one and with a lower cost. Similarly, in the second possible case, when

$$c(ip, jq) + c(ir, js) + c(iu, jv) > c(iu, jq) + c(ip, js) + c(ir, jv),$$

in (4) we replace elements (5) with (iu, jq), (ip, js) and (ir, jv). In both cases we get a contradiction that the original plan (4) is not optimal.

Remark 1. Theorem 1 can be used as a criterion for stopping the execution of Algorithm 1.

Remark 2. At iterations of Algorithm 1, it is possible to simultaneously carry out two or more three-element replacement operations.

The computational experiments carried out were carried out to determine the applicability of the new approach to solving the traveling salesman problem using the modified distribution method technology. The experiments were carried out based on the hypothesis that if the algorithm gives consistently good results for problems of size n up to the order of 20–25, then, spending time proportional to n, we will obtain an acceptable solution for large n. As a result of experiments, it turned out that this approach is realistic for obtaining an acceptable solution in a small number of iterations, no more than n. Problems

with a known optimal solution and problems generated using the algorithm from [5] were solved, when a solution is specified and a problem with this optimal solution is constructed. Algorithm 1 was used. Starting with a random route, three-element replacement operations were performed repeatedly to generate new cyclic solutions. If the new solution is better than the current best solution, it becomes the new best solution. The algorithm's calculations end when further improvements using three-element replacement operations are not possible.

Thus, many areas of the economy are developing along the path of greater automation. Algorithms with good performance are important. More advanced algorithms are necessary and relevant in cases where it is necessary to carry out analysis with large amounts of information in a short execution time. The proposed implementation of a new approach to solving the traveling salesman problem, based on an adaptation of the modified distribution method, has a number of advantages over well-known algorithms: is a more natural approach in terms of expanding the possibilities for solving the traveling salesman problem as a transport-type problem; Due to the sequential application of three-element replacement operations or the simultaneous execution of several three-element replacement operations, there is a rapid approach to an acceptable solution without large requirements for RAM.

A program for solving the traveling salesman problem using modified distribution method technology

```
% Main part of Algorithm 1
function TSP
 xv=randperm(n); yv=[xv(2:n) xv(1)]; M=Inf;  % Initial solution
          f=CelFunc(n,c,xv,yv);
          fop=f; xvop=[xv]; yvop=[yv]; iop=0;
          for it=1:It
              X(1:n,1:n)=0;
              for i=1:n X(xv(i),yv(i))=1; X(i,i)=M; end
              for np=1:Np
                 [u,v,x]=fmdr(n,X,c);
                 if puv==2*n
                   for i=1:n x(i,i)=0; end
                   break;
                 end
               end  % np
[muv,t,tm]=MatrPot(n,x,c,u,v);  % t-max muv in position (r,k)
          if t<=0  break; end
          for i=1:n muv(i,i)=-Inf; end
          [fopt,xv2,yv2,cucr]=fbsol(n,c,x,xv,yv,muv,Inf);
          if fopt<fop
              fop=fopt; xvop=[xv2]; yvop=[yv2]; iop=it;
          end
          if cucr==0 break; end
           xv=[xvop]; yv=[yvop];
            end  % it
            fprintf('\n\n The best decision fop=%d за iop=%d \n
(xvop,yvop):', fop,iop);
           for ic=1:n fprintf(' (%d,%d)', xvop(ic),yvop(ic)); end

        %     Solution search function fbsol
function [fo pt,xv2,yv2,cucr]=fbsol(n,c,x,xv,yv,muv,Inf)
          %The function corresponds to algorithm 3
           fopt=Inf; xv2(1:n)=0; yv2(1:n)=0;
           ctc=0; ccr=0; cucr=0;  % Computational process parameters

           for i=1:n
              for j=1:n
                 if i~=j && x(i,j)==0    % && muv(i,j)>0
                   for jc=1:n
                     if x(i,jc)==1 a1=i; b1=jc; break; end
                   end
                   for ic=1:n
                     if x(ic,j)==1 a2=ic; b2=j; break; end
                   end

           for ic=1:n
               if  xv(ic)~=a1  &&  yv(ic)~=b1  &&  xv(ic)~=a2  &&
```

```
yv(ic)~=b2
                        ctc=ctc+1;
                        a3=xv(ic); b3=yv(ic);
              ci(1)=a1; cj(1)=b1; ci(4)=a2; cj(4)=b1; ci(7)=a3; cj(7)=b1;
              ci(2)=a2; cj(2)=b2; ci(5)=a3; cj(5)=b2; ci(8)=a1; cj(8)=b2 ;
ci(3)=a3; cj(3)=b3; ci(6)=a1; cj(6)=b3; ci(9)=a2; cj(9)=b3;
                      sc0=c(a1,b1)+c(a2,b2)+c(a3,b3);
                      sc1=c(a2,b1)+c(a3,b2)+c(a1,b3);
                      sc2=c(a3,b1)+c(a1,b2)+c(a2,b3);
                      [msc,isc]=min([sc1 sc2]);
                      if sc0>msc && msc<Inf
                        X=[x]; xv1=[xv]; yv1=[yv];
                        for i1=1:3
                           ac=ci(i1); bc=cj(i1);
                           for j1=1:n
                              if ac==xv1(j1) && bc==yv1(j1)
                       if isc==1 xv1(j1)=ci(i1+3); yv1(j1)=cj(i1+3); end
                       if isc==2 xv1(j1)=ci(i1+6); yv1(j1)=cj(i1+6); end
                                 break;
                              end
                           end
                        end
          for ic=1:n X(xv(ic),yv(ic))=0; X(xv1(ic),yv1(ic))=1; end
                      % X after improvement
                      ccr=ccr+1; fsc=CelFunc(n,c,xv1,yv1);
                      if fsc<=fopt    % fsc - Best solution per iteration
                        fopt=fsc; xv2=[xv1]; yv2=[yv1]; cucr=cucr+1;
                      end
                    end %sc0>msc
                  end
                end % if i~=j ... muv(i,j)>0
              end  % j
            end  %  i
            if fopt<M
 fprintf('\n fopt=%d ctc=%d ccr=%d cucr=%d', fopt,ctc,ccr,cucr);
               for ig=1:n fprintf(' (%d,%d)',xv2(ig),yv2(ig)); end
            end
            if fopt>=Inf
              f=CelFunc(n,c,xv,yv); fprintf('\n f=%d ',f);
              for ig=1:n fprintf(' (%d,%d)',xv(ig),yv(ig)); end
            end
```

4.7 A New Approach to Solving the Traveling Salesman Problem

The study relates to the traveling salesman problem, which consists of finding the optimal route that passes through each of the given points once and then returns to the starting point. The goal of the traversal is to minimize a function related to the route cost, distance traveled, or time spent [140-146].

According to its mathematical formulation, the traveling salesman problem is related to transport-type problems and can be used in other situations: to optimize the data transmission route between different nodes, in creating cost-effective production, drone mission planning, to develop more advanced software systems useful for technologies such as driverless cars, and in developing software for processing large amounts of information.

In continuation of the research [141,145,146], the goal here is to develop a new computational algorithm for solving the traveling salesman problem, based on the use of the three-element replacement operation and the technology of the modified distribution method.

4.7.1 Optimality criteria

Let us present two criteria for the optimality of the solution to the traveling salesman problem. One criterion is presented as a necessary feature of optimality, and the other as a sufficient condition for the optimality of the solution.

Definition: The operation of three-element replacement is the replacement of three elements (i_1, j_1), (i_2, j_2), (i_2, j_3) of a given route with a cyclic sequence of other three elements.

Let c be the matrix of transport costs, $x_{a_1b_1}, x_{a_2b_2}, ..., x_{a_nb_n}$ - arbitrary feasible route (solution) to the traveling salesman problem. We form combinations of three unit elements $x_{i_1 j_1} = 1$, $x_{i_2 j_2} = 1$ and $x_{i_3 j_3} = 1$ in this solution and calculate

$$\text{C(i1,j1,i2,j2,i3,j3)} = c_{i_1 j_1} + c_{i_2 j_2} + c_{i_3 j_3}$$

Then for the optimal plan and any of its 3-element combinations, the following inequality must be satisfied

$$C(i_1, j_1, i_2, j_2, i_3, j_3) \le min\left(C(i_2, j_1, i_3, j_2, i_1, j_3), C(i_3, j_1, i_1, j_2, i_2, j_3) \right).$$

If the inequality is not satisfied for a given solution, then it is not optimal, since after performing a three-element replacement operation $x_{i_2 j_1} = 1$, $x_{i_3 j_2} = 1$, $x_{i_1 j_3} = 1$ or $x_{i_3 j_1} = 1$, $x_{i_1 j_2} = 1$, $x_{i_2 j_3} = 1$ (depending on the order of the term on the right-hand side of the inequality at which the minimum is achieved), and putting $x_{i_1 j_1} = 0$, $x_{i_2 j_2} = 0$, $x_{i_3 j_3} = 0$, we will get the best solution.

This three-element replacement operation is used to obtain a new feasible cyclic solution with a lower cost value, and the solution is again checked for finality.

A necessary criterion for the optimality of a solution is that it is impossible to perform a three-element replacement operation for an optimal cyclic solution. Based on the optimality criterion, it is possible to construct schemes for solving the traveling salesman problem within the framework of the technology of the modified distribution method for solving the transport problem. At the same time, its calculation stages change significantly in connection with the requirement for the cyclicality of the route and the elimination of the degeneracy of the solution in the corresponding transport problem.

Let us consider another criterion that is sufficient for the optimality of the cyclic set of $c_{i_1 j_1}, c_{i_2 j_2}, ..., c_{i_n j_n}$ in the traveling salesman problem.

Theorem 1. Let there be some vector $v = (v_1, v_2, ..., v_n)$ and the set of $c_{i_1 j_1}, c_{i_2 j_2}, ..., c_{i_n j_n}$ satisfies the v-optimality condition: $c_{i_r j_r} - v_{j_r} \le c_{i_r k} - v_k$ for all k. Then this set is a solution to the traveling salesman problem.

A sufficient criterion can be used to test a given solution for optimality. To do this, a system of inequalities $c_{i_r j_r} - v_{j_r} \le c_{i_r k} - v_k$ is constructed for each element of the $c_{i_r j_r}$. If the vector v can be determined from these inequalities, then the set $c_{i_1 j_1}, c_{i_2 j_2}, ..., c_{i_n j_n}$ is optimal, which follows from the cyclicity of the set and the equality

$$\sum_{r=1}^{n} c_{i_r j_r} = \sum_{r=1}^{n} v_{j_r} + \sum_{r=1}^{n} (c_{i_r j_r} - v_{j_r}),$$

in which on the right-hand side one sum over v is constant, and the other is minimal due to the v-minimality of the elements $c_{i_r j_r}, r = \overline{1, n}.$

4.7.2 Solution of the traveling salesman problem as a special problem of the transport type

At the initial stage of Algorithm 1, two methods were used to obtain a non-degenerate cyclic solution: using the cost matrix ms (mc method) and the evaluation matrix mo (mo method).

When using the matrix mc (mo), its elements were first ordered in ascending (descending) order, and then, based on the first element selected from the ordered sequence, the next element was selected, representing a cyclical continuation of the previous one. Such iterations are carried out until a final cyclic solution is obtained.

At the stage of calculating potentials, the constructed solution must be supplemented with fictitious zeros to eliminate degeneracy and in such a way that it is possible to complete the process of determining all *2n* potentials.

Algorithm 1.

1. Definition of the matrix $c = \{c_{ij}\}$ of order n.

2. Formation of the initial route (xv, yv), $f = \sum_{i=1}^{n} c(xv_i, yv_i)$.

3. Start of iteration of searching for the optimal solution.

4. Generating a non-degenerate solution *x* corresponding to the transport problem and consisting of *n* unit elements $x(xv_i, yv_i) = 1$ and *n-1* dummy zeros $x(xve_j, yve_j)$, $j = \overline{1,n}$. Calculation of potentials u_i and v_j, j, $j = \overline{1,n}$.

5. Definition of the evaluation matrix $muv = muv(u_i, v_j)$.

6. For each zero position (i_1, j_1) of the matrix *x*, $x_{i_1 j_1} = 0$, and with a positive value of the evaluation matrix *muv*, $muv(u_i, v_j) > 0$, calculation of two cycles (1) and (2) is performed.

7. One cycle is built starting from cell zero to cell one in a row i_1,

$$(i_1^1, j_1^1), (i_2^1, j_2^1), (i_3^1, j_3^1), ..., (i_{k-1}^1, j_{k-1}^1), (i_k^1, j_1^1) \qquad (1)$$

and another cycle starting from the zero cell in the direction of the fictitious zero in the row i_1,

$$(i_1^\varepsilon, j_1^\varepsilon), (i_2^\varepsilon, j_2^\varepsilon), (i_3^\varepsilon, j_3^\varepsilon), ..., (i_{l-1}^\varepsilon, j_{l-1}^\varepsilon), (i_l^\varepsilon, j_1^\varepsilon). \qquad (2)$$

8. Recalculation of the initial solution (xv, yv) for each of the cycles.

9. Formation of new solutions (xm_1^1, ym_1^1) and $(xm_1^\varepsilon, ym_1^\varepsilon)$.

10. Checking the cyclicality of new solutions.

11. Selecting the best cyclical solution.

12. Finding the best solution based on (xm_1^1, ym_1^1) and $(xm_1^\varepsilon, ym_1^\varepsilon)$ solutions using 3-element replacement operations.

13. Based on the best cyclic solution found, we form a new initial solution (xv, yv).

14. Go to step 3, to the next iteration..

15. Steps 3-14 are repeated a specified number of times if we do not know in advance the optimal solution to the problem being solved; when solving test

problems such that the optimal solution is known in advance, the iterations continue until the final solution is obtained.

The presented algorithm 1 is a sequential application of 3-element substitution operations to solve the traveling salesman problem. The new cyclic solution is calculated by the cycle within the framework of the modified distributive method.

Note. Theorem can be interpreted as an example in a new direction in mathematics, when it is necessary to describe the entire set of problems of a given model that have a given solution.

Conclusion. By successively applying three-element replacement operations in the current cycle of the modified distribution method, a good approximation to an acceptable solution occurs without large requirements for RAM. Computational experiments were carried out in various variants of generating the initial reference plan: using the above-described mc method, the mo method, or with the selection of a random route, which was completed to the reference plan. If the new solution is better than the current one, it becomes the initial solution. This adaptation of the exact modified distributive method proves useful and is recommended for obtaining an acceptable solution to the traveling salesman problem of sufficiently large size in the allotted time.

4.8 Construction of the traveling salesman problem with a predetermined optimal solution.

In this section we present two algorithms for constructing a traveling salesman problem with a given optimal solution. The first method is intended for the case when duality break does not occur. Such traveling salesman problems can be solved by methods for solving the choice problem. Another, more general method for constructing a traveling salesman problem with a given optimal solution shows in more detail the structure of the traveling salesman problem by decomposing the cost matrix into three component matrices: the optimality matrix, the base matrix, and the reduction matrix.

4.8.1 Algorithm for constructing the traveling salesman problem with a given optimal solution

An important issue in the study of algorithms for solving combinatorial problems is the availability of test problems with a known optimal solution. In this regard, you can use the following algorithm for generating the traveling salesman problem, for which the given route

$$(p_1, p_2, \ldots, p_n, p_1) \tag{1}$$

is optimal.

Algorithm 1.

We will present a method for constructing such a traveling salesman problem, which has a predetermined optimal solution and can be solved as efficiently as the assignment problem.

1. A square non-negative $C=\{c_{ij}\}$, $i=1,2,\ldots,n;\ j=1,2,\ldots,n$ matrix is generated of dimension from which the corresponding traveling salesman problem will be formed.

2. Along the chosen cyclic path (1) we construct a matrix x for which the elements

$$x(pi,pi+1)=1,\ i=1,2,\ldots,n-1;$$
$$x(pn,p1)=1.$$

3. We enter "fictitious" zeros ε, ε - a sufficiently small positive number such that *x* is the solution to the constrained problem

$$\sum_{i=1}^{n} x_{ij} = 1+\varepsilon,\ \ j=1,2,\ldots,n$$

$$\sum_{j=1}^{n} x_{ij} = 1+\varepsilon,\ \ i=1,2,\ldots,n$$

4. By positive *2n-1* matrix elements *x*, of which n are equal to "*1*" and *n-1* "fictitious" zeros, we make equations

$$u_i + v_j = c_{ij}$$

5. We find the potentials u_i and v_j by solving the compiled system

6. Compiling a new c^* matrix,

$$c^* = \begin{cases} c_{ij}, & x(i,j) > 0 \\ u_i + v_j + \alpha_{ij}, & x(i,j) = 0 \end{cases} \tag{2}$$

where α_{ij} - any non-negative numbers.

7. For each set of α_{ij} values, we obtain one of the traveling salesman problems with the c^* cost matrix for which the

$$p^* = (p_1, p_2, ..., p_n, p_1)$$

is optimal.

Example. Consider the traveling salesman problem [22, p.141] with the cost matrix

$$c = \begin{pmatrix} M & 3 & 93 & 13 & 33 & 9 & 57 \\ 4 & M & 77 & 42 & 21 & 16 & 34 \\ 45 & 17 & M & 36 & 16 & 28 & 25 \\ 39 & 90 & 80 & M & 56 & 7 & 91 \\ 28 & 46 & 88 & 33 & M & 25 & 57 \\ 3 & 88 & 18 & 46 & 92 & M & 7 \\ 44 & 26 & 33 & 27 & 84 & 39 & M \end{pmatrix}$$

The cost of the optimal path, for example *(1, 4 , 6, 7, 3, 5, 2, 1*), is *126*. For a given solution, it may turn out that the evaluation matrix will have several positive elements, despite the fact that the solution is optimal for the traveling salesman problem with matrix ***c***.

Consider the general case in the sense that an arbitrary path is chosen, for example, (1, 2, 5, 7, 3, 4, 6, 1). Let us construct the traveling salesman problem for which this path is optimal. To do this, using the technology of solving the transport problem to the traveling salesman problem [146], we will compose a feasible solution, for example, in the form of a matrix x,

$$x = \begin{pmatrix} 0 & 1 & 0 & 0 & \varepsilon & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & \varepsilon \\ 0 & 0 & 0 & 1 & 0 & \varepsilon & 0 \\ \varepsilon & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \varepsilon & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \varepsilon & 0 & 0 & 0 \end{pmatrix}$$

in which the elements "1" correspond to the transitions of the selected path (1, 2, 5, 7, 3, 4, 6, 1) and small positive numbers ε are “fictitious” zeros. These zeros must be chosen so that the resulting solution is non-degenerate, which ensures the possibility of determining all potentials.

To calculate the potentials u and v corresponding to the chosen solution, we use all *2n-1* positive elements of the matrix *x*,

$$u=(\ 0,\ -12,\ -35,\ -56,\ 11,\ -92,\ -44),$$

$$v=(\ 95,\ 3,\ 77,\ 71,\ 33,\ 63,\ 46).$$

The required matrix, according to (2) and the evaluation matrix, can be as follows:

$$c^* = \begin{pmatrix} M & 3 & 93 & 71 & 33 & 63 & 57 \\ 83 & M & 77 & 59 & 21 & 51 & 34 \\ 60 & 17 & M & 36 & 16 & 28 & 25 \\ 39 & 90 & 80 & M & 56 & 7 & 91 \\ 106 & 46 & 88 & 82 & M & 74 & 57 \\ 3 & 88 & 18 & 46 & 92 & M & 7 \\ 51 & 26 & 33 & 27 & 84 & 39 & M \end{pmatrix}$$

The justification for the fact that the traveling salesman problem with the $c^* = \left(c_{ij}^*\right)$ matrix has the chosen route (*1, 2, 5, 7, 3, 4, 6, 1)* as a solution is the non-positivity of the elements of the evaluation matrix

$$\Delta_{ij} = u_i + v_j - c_{ij}^*$$

for non-basic elements of the matrix x and their equality to zero for basic elements.

It follows from the presentation of the algorithm and the example that for any traveling salesman problem it is possible to construct a new problem with the same optimal solution, but such that all elements of the c^* evaluation matrix are non-negative. This fact can be used to solve the traveling salesman problem by the potential method [146].

Further, the Appendix describes the function in the MATLAB environment for calculations according to the algorithm and a demo example.

The results of the research and the developed software confirm the efficiency of constructing a problem with a given optimal cyclic solution. They can be used to study the structure of a set of traveling salesman problems with a given optimal solution, to test methods for solving traveling salesman problems using the potentials method.

Construction of the traveling salesman problem with a given optimal solution is carried out using the GenerPSD function:

```
function GenerTSP
   [n,c]=FormMatrC(a,b,a1,b1,M); % Entering initial data a,b,a1,b1 for
                                 % the formation of an auxiliary matrix
                                 % c of dimensions n*n, M is a sufficiently
                                 % large positive number

   xv=randperm(n); yv=[xv(2:n) xv(1)];
        % (xv,yv) - cyclic substitution of order n
   fprintf('\n\n  (xv,yv):');
   for i=1:n fprintf(' (%d,%d)',xv(i),yv(i)); end
   X(1:n,1:n)=0;
   for i=1:n X(xv(i),yv(i))=1; X(i,i)=M; end
   XP(1:n,1:n)=0;

   for np=1:10
     [u,v,x,puv]=FMDR2(n,X,c); % Construction of a feasible solution x
                               % and calculation of potentials u and v

     if puv==2*n  % puv a sign of obtaining a reference solution
                  % and the possibility of calculating all
                  % potentials u and v
       for i=1:n x(i,i)=0; end
       break;
     end
   end  % np

   muv=MatrPot(n,x,c,u,v);  % muv scoring matrix calculation function,
                            % muv=ui+vj-cij
   C=c;
   for i=1:n
     for j=1:n
       if i~=j && x(i,j)==0 && muv(i,j)>0
         alfa=floor(10*rand)+1;
         C(i,j)=u(i)+v(j)+alfa; % Definition of matrix C for the
                                % traveling salesman problem with
                                % optimal solution (xv,yv)
       end
     end
   end

   [u,v]=DetPot(n,x,C);
   muv=MatrPot(n,x,C,u,v); % Calculation of the muv matrix for matrix C
                           % for the purpose of checking whether all muv
                           % elements are non-positive. Then (xv,yv)
                           % represents the optimal solution for the problem
                           % with matrix C.

function [n,c]=FormMatrC(a,b,a1,b1,M)  % Construction of an auxiliary
square
```

```
                                % matrix c of order n
n=a+floor(b*rand);
for ic=1:n
   for jc=1:n
      if ic==jc
         c(ic,jc)=M; else c(ic,jc)=a1+floor(b1*rand);
      end
   end
end

function [u,v,x,puv]=FMDR2(n,X,c)   % Formation of a feasible solution x
                                % and calculation of potentials u and
  u(1:n)=0; v(1:n)=0; cp=1; i1=1; ic=1; r=n-2;
  iu(1:n)=0; jv(1:n)=0; iu(1)=1; is(1:n)=0; js(1:n)=0;
  x=[X];
  for jc=1:n
     if x(1,jc)==1 v(jc)=c(1,jc); cp=cp+1; j1=jc; is(i1)=i1;
        break;
     end
  end

 for k=1:n-1
   s=0;
   for ic=1:n
     if x(ic,j1)==0 && is(ic)==0 s=s+1; end
   end
     h=floor(s*rand)+1; r=0;
   for ic=1:n
     if x(ic,j1)==0 && is(ic)==0 r=r+1;
        if r==h
          u(ic)=c(ic,j1)-v(j1); i1=ic; cp=cp+1; iu(ic)=ic;
          x(ic,j1)=0.01; is(ic)=ic; js(j1)=j1;
          break;
        end
     end
   end  % ic

   for jc=1:n
     if x(i1,jc)==1 && jc~=j1
        v(jc)=c(i1,jc)-u(i1); j1=jc; cp=cp+1; jv(jc)=jc;
        break;
     end
   end
 end  %  k

 puv=1; for ic=1:n
          if iu(ic)>0 puv=puv+1; end
          if jv(ic)>0 puv=puv+1; end
        end
```

```
  if puv==2*n fprintf('\n All potentials are defined ');
  else pt=2*n-puv;
     fprintf('\n Not all potentials are defined ');
  end

function muv=MatrPot(n,x,c,u,v)   % Definition of muv score matrix
  muv=zeros(n);   % muv(i,j)=u(i)+v(j)-c(i,j);
  for i=1:n
    for j=1:n
       s=u(i)+v(j)-c(i,j); muv(i,j)=s;
    end
  end
function [u,v]=DetPot(n,x,c)  % Determination of potentials u and v
  u(1:n)=0; v(1:n)=0; cp=1; i1=1; ic=1;
  iu(1:n)=0; jv(1:n)=0; iu(1)=1;
  for jc=1:n
     if x(1,jc)==1 v(jc)=c(1,jc); cp=cp+1; j1=jc; jv(jc)=jc;
        break;
     end
  end

  for k=1:n-1
    for ic=1:n
      if x(ic,j1)>0 && ic~=i1
        u(ic)=c(ic,j1)-v(j1); i1=ic; cp=cp+1; iu(ic)=ic;
        break;
      end
    end  % ic

    for jc=1:n
      if x(i1,jc)==1 && jc~=j1
        v(jc)=c(i1,jc)-u(i1); j1=jc; cp=cp+1; jv(jc)=jc;
        break;
      end
    end
  end  %  k
   puv=0; for ic=1:n
            if iu(ic)>0 puv=puv+1; end
            if jv(ic)>0 puv=puv+1; end
          end
   if puv==2*n fprintf('\n All potentials are defined ‘,puv);
   else pt=2*n-puv;
      fprintf('\n Number of undefined potentials pt=%d',pt);
   end
   fprintf('\n u:'); for i=1:n fprintf(' %d', u(i)); end
   fprintf('\n v:'); for i=1:n fprintf(' %d', v(i)); end
```

An example of constructing the traveling salesman problem with a given optimal solution. using the GenerTSP function in MATLAB.

1. Definition of an arbitrary non-negative matrix c with dimension n=7

$$c = \begin{pmatrix} M & 15 & 24 & 22 & 20 & 21 & 24 \\ 20 & M & 15 & 24 & 20 & 18 & 15 \\ 22 & 19 & M & 22 & 18 & 15 & 17 \\ 22 & 24 & 18 & M & 22 & 23 & 17 \\ 22 & 16 & 20 & 18 & M & 15 & 15 \\ 22 & 22 & 16 & 20 & 15 & M & 19 \\ 22 & 21 & 23 & 22 & 22 & 19 & M \end{pmatrix}$$

2. A cyclic solution is given

(xv,yv): (7,4) (4,6) (6,1) (1,3) (3,5) (5,2) (2,7)

3. Determination of potentials u and v:

u=(0, 5, -1, 9, -4, 9, 3)

v=(13, 20, 24, 19, 19, 14, 10)

4. Matrix of estimates $muv = u_i + v_j - c_{ij}$:

$$muv = \begin{pmatrix} -M & 5 & 0 & -3 & -1 & -7 & -14 \\ -2 & -M & 14 & 0 & 4 & 1 & 0 \\ -10 & 0 & -M & -4 & 0 & -2 & -8 \\ 0 & 5 & 15 & -M & 6 & 0 & 2 \\ -13 & 0 & 0 & -3 & -M & -5 & -9 \\ 0 & 7 & 17 & 8 & 13 & -M & 0 \\ -6 & 2 & 4 & 0 & 0 & -2 & -M \end{pmatrix}$$

5. The result of the construction is the following matrix C:

$$C = \begin{pmatrix} M & 24 & 24 & 22 & 20 & 21 & 24 \\ 20 & M & 37 & 24 & 34 & 27 & 15 \\ 22 & 19 & M & 22 & 18 & 15 & 17 \\ 22 & 33 & 36 & M & 37 & 23 & 26 \\ 22 & 16 & 20 & 18 & M & 15 & 15 \\ 22 & 34 & 40 & 30 & 32 & M & 19 \\ 22 & 29 & 36 & 22 & 22 & 19 & M \end{pmatrix}$$

6. Checking the optimality of the chosen cyclic solution for the traveling salesman problem with matrix C is performed using the corresponding evaluation matrix:

$$muv = \begin{pmatrix} -M & -4 & 0 & -3 & -1 & -7 & -14 \\ -2 & M & -8 & 0 & -10 & -8 & 0 \\ -10 & 0 & M & -4 & 0 & -2 & -8 \\ 0 & -4 & -3 & M & -9 & 0 & -7 \\ -13 & 0 & 0 & -3 & M & -5 & -9 \\ 0 & -5 & -7 & -2 & -4 & M & 0 \\ -6 & -6 & -9 & 0 & 0 & 2 & -M \end{pmatrix}$$

All elements of the evaluation matrix for *C* are non-positive. Therefore, the cyclic solution chosen in Section 2 is optimal for the traveling salesman problem with cost matrix *c*.

4.8.2 General case of constructing a traveling salesman problem with a given optimal solution using the function c=NovGenerTSP(n,xo,yo,mc,k)

A more general way to construct a traveling salesman problem with a given optimal solution is implemented by Algorithm 2.

Algorithm 2 is based on the representation of the cost matrix as a decomposition into components. The cost matrix **c** is represented as the sum of three matrices: *co* is the optimality matrix, *cb* is the base matrix, and *cr* is the reduction matrix.

Algorithm 2.

Step 1. Generate a cyclic substitution *(xo,yo)* that corresponds to the optimal route

$$(i_1, j_1),(i_2, j_2),(i_3, j_3),...,(i_{n-1}, j_{n-1}),(i_n, j_n).$$

Step 2. Formation of the optimality matrix *co*, in which the elements in positions (*xo(k),yo(k)*) are non-zero, as well as the diagonal elements,

$$co(i, j) = \begin{cases} -a, \ i = i_k, j = j_k, \ k = \overline{1,n} \\ Inf, \ i = j, \\ 0, \ in\ other\ cases \end{cases}$$

where *a* is a positive integer constant.

Step 3. Generate the base matrix *cb* with elements

$$cb(i, j) = \begin{cases} 0, \ for\ i = i_k, j = j_k\ or\ i = j, \\ b_{ij}, \ in\ other\ cases \end{cases}$$

where b_{ij} are integer non-negative random numbers not less than *a*.

Step 4. We define two random positive integer vectors p and q and a reduction matrix cr,

$$cr(\mathrm{i}, j) = \begin{cases} p_i + q_j, \ i \neq j \\ 0, \ i = j \end{cases}$$

Step 5. We construct the cost matrix c of the traveling salesman problem with a given optimal solution (xo,yo),

$$c = co + cb + cr.$$

Theorem. Traveling salesman problem with cost matrix $c = co + cb + cr$, constructed according to Algorithm 2, has an optimal solution (xo,yo).

Proof. From the construction of the matrix c it follows that, after reduction by rows and columns, in the matrix $co + cb = \left\{ c_{ij} - p_i - q_j \right\}$ in positions ($xo(i)$,$yo(i)$), $i = \overline{1,n}$, there are zeros, and in the remaining positions there are non-negative numbers. By construction, the sequence ($xo(i)$,$yo(i)$) forms a cyclic route. Therefore ($xo(i)$,$yo(i)$), $i = \overline{1,n}$, is the optimal route of the traveling salesman problem with matrix c.

The general case of the formation of traveling salesman problems with a given optimal solution is carried out by the function *c=NovGenerTSP(n,xo,yo,mc,k)*, n is the dimension of the efficiency matrix of the corresponding problem, *(xo,yo)* is its given optimal solution, *mc* is the upper bound of the matrix c, k is the number of iterations of obtaining the reduction matrix.

```
% Formation of TSP with a given optimal solution (xo,yo)
% Параметры mc и k
function c=NovGenerTSP(n,xo,yo,mc,k)
  c=zeros(n);
  for i=1:n c(xo(i),yo(i))=-1; c(i,i)=Inf; end
  for i=1:n
     for j=1:n
        if i~=j && c(i,j)==0 alfa=floor(mc*rand)+1; c(i,j)=alfa; end
     end
  end

  for h=1:k
    p=randperm(n); q=randperm(n);
    for i=1:n
      for j=1:n
        if i~=j c(i,j)=c(i,j)+p(i)+q(j); end
      end
    end
  end
```

In conclusion, we note that the monograph presents mainly the author's developments on methods, algorithms and programs for solving integer linear programming and combinatorial optimization problems. The field of research in these areas of discrete optimization is quite extensive. The list of references [1-155] provides sources (in three languages: eng - in English [63-155], rom - in Romanian [49-62], rus - in Russian [1-48]), which reflect the main trends of research in discrete otimization.

REFERENCES

rus

1. Arkhipova T. T., Sergienko I. V. The method of feasible directions and the method of decay vector in integer programming. Cybernetics. 1975. No. 5. Pp. 125-128.

2. Bystrov V. A., Shcherbina O. A. On the efficiency of solving one class of discrete programming problems using a local algorithm. Cybernetics. 1987. No. 3. Pp. 96-101.

3. Emelichev V. A. On the Theory of Discrete Optimization. Reports of the USSR Academy of Sciences. 1971. Vol. 198, No. 2. P. 273-276.

4. Emelichev V. A., Kovalev M. M. Polyhedral aspects of discrete optimization // Cybernetics. 1982. No. 6. P. 54-62.

5. Emelichev V. A., Komlik V. I. Method of constructing a sequence of plans for solving discrete optimization problems. Moscow: Science, 1981. 208 p.

6. Finkelstein Yu. Yu. Approximate methods and applied problems of discrete programming. Moscow: Nauka, 1970, 2000. 264 p.

7. Grishukhin V. P. Branch and Bound Algorithms in Problems with Boolean Variables, Evaluation of Their Efficiency. Economics and Mathematical Methods. 1976.1. Vol. 12, No. 4. P. 757-766.

8. Hu T. Integer Programming and Network Flows. Moscow: Mir, 1974. 519 p.

9. Ivanov B. N. Discrete Mathematics. Algorithms and Programs: Textbook. Moscow: Basic Knowledge Laboratory, 2003. 288 p.

10. Kanaeva N. N., Shcherbina O. A. On a Modified Local Algorithm for Discrete Programming Problems // Russ. Comput. Math. and Mathematical Physics. 1986. Vol. 26, No. 2. P. 263-275.

11. Khachaturov V. R., Veselovsky V. E., Zlotov A. V. et al. Combinatorial Methods and Algorithms for Solving Large-Scale Discrete Optimization Problems. M.: Nauka, 2000. 354 p.

12. Kovalev M. M., Pir'yanovich V. A. Locally stochastic algorithms of discrete optimization (experiments, experience, applications). Cybernetics. 1982. No. 1. P. 108-114.

13. Kovalev M. M. Discrete optimization (integer programming). Moscow: Editorial URSS, 2003. 192 p.

14. Korbut A. A., Sigal I. Kh., Finkelstein Yu. Yu. Branch and bound method (review of the theory of algorithms, programs and applications). Math. Operationsforsch. Statist. Ser.Optimization. 1977. 8, No. 2. P. 253-280.

15. Korbut A. A., Finkelstein Yu. Yu. Discrete programming. Moscow: Nauka, 1969. 368 p.

16. Kochetov Yu. A. Computational capabilities of local search in combinatorial optimization. Journal of Computational Mathematics and Mathematical Physics. 2008. Vol. 48, Pp. 747-764.

17. Kuzyurin N. N. Probabilistic approximate algorithms in discrete optimization. Discrete analysis and operations research, series 2. 2002. Vol. 9, No. 2. Pp. 97-114.

18. Leontiev V. K. Discrete extremal problems // Results of science and technology of VINITI. Series: Probability theory, mathematical statistics, theoretical cybernetics. Moscow: VINITI. 1979. Issue 16. Pp. 39-101.

19. Leontiev V. K. Discrete optimization. Journal of Computational Mathematics and Mathematical Physics. 2007. Vol. 47, No. 2. Pp. 338-352.

20. Mikhalevich V. S., Volkovich V. L., Voloshin A. F., Poznyakov Yu. M. Algorithms for sequential analysis and screening of variants in discrete optimization problems. Cybernetics. 1980. No. 3. P. 76-85.

21. Mikhalevich V. S., Sergienko I. V. et al. The DISPRO-3 software package: purpose, classes of problems to be solved, system and algorithmic support. Cybernetics. 1985. No. 1. P. 56-71.

22. Papadimitriou H., Steiglitz K. Combinatorial Optimization. Algorithms and Complexity. Moscow: Mir, 1985. 512 p.

23. Romakin M.I. Optimization of production planning. Moscow, Finance and Statistics, 1981. 109 p.

24. Sergienko I. V., Kaspshitskaya M. F. Models and Methods for Solving Combinatorial Optimization Problems on a Computer. Kyiv: Naukova Dumka, 1981. 288 p.

25. Sergienko I. V. Mathematical Models and Methods for Solving Discrete Optimization Problems. Kyiv: Naukova Dumka, 1988. 472 p.

26. Sergienko I. V., Lebedeva T. T., Roshchin V. A. Approximate methods for solving discrete optimization problems. Kyiv: Naukova Dumka, 1980. 276 p.

27. Sergienko I. V., Shilo V. P. Discrete Optimization Problems: Complex Tasks, Basic Approaches to Their Solution. Cybernetics and Systems Analysis. 2006. No. 4. p. 3-25.

28. Terzi D. Iterative process of approximate solution of problems with Boolean variables. Optimization, 1973, 10(27).

29. Terzi D. Hermite algorithm and solution of discrete programming problems. Optimization, 1973, 11(28).

30. Terzi D. On some variants of the cutting method. Optimization, 1973, 11(28).

31. Terzi D. On one cutting. In the collection Optimization of territorial and sectoral systems methods for solving economic problems, part III. Novosibirsk, 1973, p.100-102.

32. Terzi D. On one feature of some integer linear programming algorithms. Controlled systems. Novosibirsk, IM SB RAS USSR, 1974, p. 68-72.

33. Terzi D. Solution of a system of linear equations in integers VINITI. State Fund of Algorithms and Programs of the USSR. Algorithms and Programs. Information Bulletin, 1984, 5(62), P007553.

34. Terzi D. Program for processing geographical information (construction of Voronoi diagrams). Deposited at VNTICenter, registration number 50920000/45 dated 13.10.1992.

35. Terzi D. On the aggregation of equations. In the collection Applied Mathematics and Programming, issue 11. Chisinau, Shtiinta, 1974.

36. Terzi D. et al. Integer model of the problem of rationalization of the network of comprehensive schools and its solution. In the collection Mathematical modeling of planning and control processes. Mathematical researches, issue 60. Chisinau, Shtiinta, 1980.

37. Terzi D. Study of discrete programming algorithms. In the collection Mathematical modeling of planning and control processes. Mathematical researches, issue 60. Chisinau, Shtiinta, 1980.

38. Terzi D. Approximate solution of integer linear programming problems. In the collection Models and methods for solving planning and control problems. Mathematical researches, issue 39. Chisinau, Shtiinta, 1982.

40. Terzi D., et al. On one problem of agricultural crop placement. In the collection Models and algorithms for solving planning and control problems. Mathematical researches, issue 68. Chisinau, 1982.

41. Terzi D. On approximate solution of integer linear programming problem. In: Mathematical models and methods of optimization of economic systems. Mathematical researches, issue 72. Kishinev, Shtiinta, 1983.

42. Terzi D. Vector generators in discrete optimization problems. In: Mathematical models and methods of optimization. Mathematical researches, issue 82. Chisinau, Shtiinta, 1985.

43. Terzi D. The problem of minimizing the function associated with the productivity of natural complexes. In the collection of VII All-Union Conference "Problems of Theoretical Cybernetics". Abstracts of reports, part II. Irkutsk, USSR Academy of Sciences, Institute of Applied Mathematics, IrVC, Ministry of Higher Education of the RSFSR, Moscow State University, IrSU, 1985.

44. Terzi D. On the efficient solution of a special problem of integer linear programming. In the collection of Theoretical Problems of Cybernetics, part 1. Saratov, Saratov State University, 1986.

45. Terzi D. Study of the efficiency of integer linear programming algorithms. In the collection of Software Systems for Solving Optimal Planning Problems. Sixth All-Union Symposium (Pushchino, June 2-10, 1980). Central Economics and Mathematics Institute of the USSR Academy of Sciences, 1980.

46. Terzi D., et al. Creation of a GIS for solving problems of optimization and forecasting the state of the natural environment. In the book. Integrated monitoring and practice (Abstracts of reports of the All-Union symposium Integrated monitoring, optimization and forecasting of the state of the natural environment, 1991. Upper Volga region). Moscow, USSR Academy of Sciences, 1991.

47. Terzi D. Modeling ARIMA schemes for economic forecasting. Dezvoltarea economică în contextul aspiraţiei de integrare europeană. Perspective şi realizări. Conferinţa ştiinţifică internaţională, 23-24 Octombrie 2009.

48. Volkovich S. V., Voloshin A. F. On a General Scheme of the Method of Sequential Analysis and Screening of Variants // Cybernetics. 1978. No. 5. P. 98-105.

rom

49. Terzi D. Previziunea Economică. Practicum, Chişinău, CEP USM, 2015.

50. Terzi D. Brăilă A. Previziunea Economică. Suport de curs, Chişinău, CEP USM, 2013.

51. Terzi D.. Branch and bound pentru rezolvarea problemelor de programare discretă. Paradigme moderne în dezvoltarea economiei naţionale şi mondiale. Cu ocazia aniversării a 65 de ani de la fondarea Facultăţii de Ştiinţe Economice, 2018. Chişinău. ISBN 978-9975-142-57-1.

52. Terzi D. O abordare privind rezolvarea problemei comis-voiajorului cu ajutorul metodei potenţialelor. Studia Universitatis Moldaviae (Seria Ştiinţe Exacte şi Economice), Nr. 2(122), 2019. ISSN 1857-2073 /ISSNe 2345-1033.

53. Terzi D. Particularităţile de creare şi aplicare a reţelelor neuronale pentru previziune economică. Integrare prin cercetare şi inovare.Ştiinţe sociale, vol.1, 2014. Chisinau, Republica Moldova.

54. Terzi D. Construirea unei soluţii admisibile pentru problema comis-voiajorului şi verificarea condiţiei de optimalitate. Paradigme moderne în dezvoltarea economiei naţionale şi mondiale, 2019. Chişinău, ISBN 978-9975-142-57-1.

55. Terzi D. O soluţie aproximativă a problemei comis-voiajorului folosind un set de euristic. Studia Universitatis Moldaviae (Seria Ştiinţe Exacte şi Economice), Nr. 2(132), 2020. ISSN 1857-2073 /ISSNe 2345-1033.

56. Terzi D. Tehnologia rețelelor neuronale de clasificare a obiectelor. Integrare prin cercetare și inovare. Ştiințe ale naturii și exacte. Ştiințe economice. 2017. Chisinau, Republica Moldova. ISBN 978-9975-71-701-4.

57. Terzi D. Modelarea reţelei neuronale ca o problemă de optimizare şi prognoză. Creşterea competitivităţii economice în contextul formării societăţii bazate pe cunoaştere 28-29 octombrie 2016, Ediţia a XI-a a Conferinţei Ştiinţifice Internaţionale, CEP USM Chişinău 2016, 0,55 c.a.

58. Terzi D. Statistical methods of discrete optimization. Creşterea economică în condiţiile internaţionalizării. I-a conferinţă ştiinţifică internaţională. Institutul de Economie, Finanţe şi Statistică, Academia de Studii Economice din Moldova. Chişinău, 2006.

59. Terzi D. Particularităţile de creare şi aplicare a reţelelor neuronale pentru previziune economică, Integrare prin Cercetare şi Inovare. Conferinţa ştiinţifică naţională cu participare internaţională 10 - 11 noiembrie 2014 Rezumate ale comuinicărilor Ştiinţe sociale Chişinău, 2014.

60. Terzi D. Tehnologia reţelelor neuronale de clasificare a obiectelor. Integrare prin cercetare şi inovare. Conferinţa ştiinţifică naţională cu participare inernaţională, 9-10 noiembrie 2017, 286-290.

61. Terzi D. Modelarea reţelei neuronale ca o problemă de optimizare şi prognoză. Creşterea competitivităţii economice în contextul formării societăţii bazate pe cunoaştere 28-29 octombrie

62. Terzi D. Program pentru calculator “Applicarea tehnologiei metodei de distribuție modificate pentru a rezolva problema comis-voiajorului”. Adeverința privind înscrierea obiectelor dreptului de autor și ale drepturilor conexe. Seria PC Nr. 7913 din 03.05.2024. Republica Moldova Agenția de Stat pentru Proprietatea Intelectuală. IDNP: 0980912012015553. Data înr.: 04.04.2024. 8 p.

eng

63. Abdel-Basset M., Abdel-Fatah L., Sangaiah A. K. Metaheuristic algorithms: A comprehensive review. In: Sangaiah A K, Zhang Z, Sheng M, eds. Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications. New York: Academic Press, 2018, 185–231.

64. Applegate D.L., Bixby R., Chvátal V., and Cook W.J. The Traveling Salesman Problem: A Computational Study. Princeton University Press, 2007.

65. Applegate D., Bixby R., Chvatal V., and Cook W. CONCORDE TSP Solver. URL: http://www.tsp.gatech.edu/concorde.html (дата обращения 23.08.2010).

66. Ascheuer N., Jünger M., Reinelt G. A Branch-and-Cut Algorithm for the Asymmetric Traveling Salesman Problem with Precedence Constraints. Computational Optimization and Applications 17(1),61-84(2000)doi:10.1023/A:1008779125567.

67. Asadpour A., Goemans M.X., Madry A., Gharan S.O., Saberi A. An o (log n/log log n)-approximation algorithm for the asymmetric traveling salesman problem. Operations Research 65(4) (2017) 1043–1061.

68. Aarts E., Lenstra J. K. (eds.) Local Search in Combinatorial Optimization. New York: Wiley, 1997. 512 c.

69. Ahuja R. K., Magnanti T. L., Orlin J. B. Network Flows: Theory, Algorithms, and Applications. Englewood Cliffs: Prentice-Hall, 1993. 846 p.

70. Ahr D., Reinelt G. A Tabu Search Algorithm for the Min-Max k-Chinese Postman Problem. Computers and Operations Research 33(12), 3403-3422 (2006).

71. Atamturk A., Savelsbergh M. W. P. Integer-programming software systems. Annals of Operations Research. 2005. V. 140. P. 67-124.

72. Ausiello G., Crescensi P., Gambosi G., Kann V., Marchetti-Spaccamela A., Protasi M. Complexity and Approximation (combinatorial optimization problems and their approximability properties). Berlin: Springer-Verlag, 1999. 524 p.

73. Bachem A., Euler R. Recent trends in combinatorial optimization. Operations Research Spektrum. 1984. V. 6. P. 1-22.

74. Beale E. M. L. Branch and bound methods for mathematical programming systems. Ann. Discrete Math., 1979. V. 5. P. 201-219.

75. Borgwardt K.H., Gaffke N., Jünger M., Reinelt G. Computing the Convex Hull in the Euclidean Plane in Linear Expected Time. In: Gritzmann, P., Sturmfels, B.: The Victor Klee Festschrift. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 4, pp. 91-107. American Mathematical Society (1991).

76. Chen Liuhong, Xiong Meixin, Ming Ju, He Xiaoming. Efficient mini-batch stochastic gradient descent with centroidal Voronoi tessellation for PDE-constrained optimization under uncertainty. Physica D 467, Article ID 134216, 14 p. (2024).

77. Christof T., Jünger M., Reinelt G. A Complete Description of the Traveling Salesman Polytope on 8 Nodes. Operations Research Letters 10(9), 497-500 (1991).

78. Crama Y, Kolen A W J, Pesch E J. Local Search in Combinatorial Optimization. Berlin-Heidelberg: Springer, 199.

79. Espejo Miranda et al. A new formulation and branch-and-cut method for single-allocation hub location problems. Comput. Oper. Res. 155, Article ID 106241, 16 p. (2023).

80. Floudas C. A. Nonlinear and mixed-integer optimization: fundamentals and applications. Oxford: Oxford Univ.Press, 1996. 462 p.

81. Floudas C.A. Deterministic Global Optimization Theory. Algorithms and Applications.

82. Gendron B., Crainic T. G. Parallel branch and bound algorithms: Survey and synthesis. Operations Research. 1994. V. 42. P. 1042-1066.

83. Genetic algorithms and simulated annealing / L. Davis (ed.). Los Altos: Morgan Kauffmann. 1987. 216 p.

84. Gens B. V., Levner E. V. Complexity of approximation algorithms for combinatorial problems: a survey. Sigact News. 1980. V. 12, N 3. P. 52-65.

85. Geoffrion A. M. Lagrangean relaxation for integer programming. Mathematical Programming Study. 1974. V. 2. P. 82-114.

86. Glover F. and Laguna M. Tabu Search. Boston: Kluwer, 1997. 382 p.

87. Gorry G. A., Shapiro J. F., Wolsey L. A. Relaxation methods for pure and mixed integer programming problems. Management Science. 1972. V. 18, N 5. P. 229-239.

88. Grama A., Kumar V. Parallel search algorithms for discrete optimization problems // ORSA Journal on Computing. 1995. V. 7. P. 365-385.

89. Groetschel M., Lovasz L., Schrijver A. Ellipsoid method and its consequences in combinatorial optimization. Combinatorica. 1981. V. 1. P. 169-197.

90. Grótschel M., Lovász L., Schrijver A. Geometric Algorithms and Combinatorial Optimization. New York: Springer-Verlag, 1988. 362 p.

91. Grötschel M., Jünger M., Reinelt G. Optimal Triangulation of Large Real World Input-Output Matrices. Statistische Hefte 25(1), 261-295 (1984). doi:10.1007/BF02932410

92. Hall M. Jr. Combinatorial theory. Blaisdell Publishing Company. Waltham (Massachusetts) Toronto London, 1967.

93. Hadley G. Nonlinear and Dynamic Programming. Addison Wesley publishing company, INC. Reading, Massachusetts * Palo alto * London, 1964.

94. Hoffman K., Padberg M. LP-based combinatorial problem solving. Annals of Operations Research. 1985. V. 4. P. 145-194.

95. Hooker J. Logic-based methods for optimization: combining optimization and constraint satisfaction. John Wiley, 2000. 495 p.

96. Ibaraki T. On the computational efficiency of branch-and-bound algorithms. Journal of the Operations Research Society of Japan. 1977. V. 20. P. 16-35.75. Cerf, Max Optimization techniques II. Discrete and functional optimization. Translated from the French.

97. Current Natural Sciences. [s.l.]: Science Press; Les Ulis: EDP Sciences (ISBN 978-2-7598-3163-0/pbk; 978-2-7598-3165-4/ebook). 464 p. (2023).

98. Ikebe Yoshiko, Masuda Satoru, Okuno Takayuki. Mixed-integer DC programming based algorithms for the circular packing problem. J. Oper. Res. Soc. Japan 66, No. 3, 153-175 (2023).

99. Jünger M., Thienel S. The ABACUS system for branch and cut and price algorithms in integer programming and combinatorial optimization. Software Practice and Experience. 2000. V. 30. P. 1325-1352.

100. Jünger M., Reinelt G. (eds.): Facets of Combinatorial Optimization, Festschrift for Martin Grötschel.Springer (2013) ISBN 978-3-642-38188-1.

101. Jünger M., Liebling T.L., Naddef D., Nemhauser G.L., Pulleyblank W.R., Reinelt G., Rinaldi G., Wolsey L.A. (eds.): 50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art. Springer, Heidelberg (2010).

102. Jünger M. Reinelt G., Zepf D. Computing Correct Delaunay Triangulations. Computing 47(1), 43-49 (1991).

103. Jung M. Relaxations and Approximations for Mixed-Integer Optimal Control. Dissertation, Universität Heidelberg (2014).

104. Kloks T. Treewidth: Computations and Approximations. New York: Springer, 1994. 209 p.

105. Korte B. Approximative algorithms for discrete optimization problems // P. L. Hammer, E. L. Johnson, B. H. Korte (Eds.): Discrete Optimization I. Annals of Discrete Mathematics. Amsterdam, New York, Oxford: North Holland. 1979. V. 4. P. 85-120.

106. Korte B., Vygen J. Combinatorial Optimization: Theory and Algorithms (Algorithms and Combinatorics, v.21), New York, 2018.

107. Laporte G. The traveling salesman problem: An overview of exact and approximate algorithms. European Journal of Operational Research 59(2) (1992) 231–247

108. Lo K.M., Yi W.Y., Wong P.K., Leung K.S., Leung Y., Mak S.T. A genetic algorithm with new local operators for multiple traveling salesman problems. International Journal of Computational Intelligence Systems 11(1) (2018) 692–705.

109. Ladányi L., Ralphs T.K., and Trotter L.E. Branch, cut, and price: Sequential and parallel // Computational Combinatorial Optimization / D. Naddef, M. Jünger (eds.). Berlin: Springer, 2001. P. 223-260.

110. Linderoth J. T., Ralphs T. K. Noncommercial software for mixed-integer linear programming. Integer Programming: Theory and Practice / J. Karlof (ed). CRC Press Operations Research Series, 2005. P. 253-303.

111. Liu Xin, Sun Jianyong, Xu Zongben. Learning to sample initial solution for solving 0-1 discrete optimization problem by local search. Sci. China, Math. 67, No. 6, 1317-1340 (2024).

112. Maffioli F. Complexity of combinatorial optimization algorithms and the challenge of heuristics // Combinatorial Optimization / N. Christofides, A. Mingozzi, P. Toth, C. Sandi (eds.). Chichester, New York, Brisbane, Toronto: Wiley. 1979. P. 107-130.

113. Martin A. General mixed integer programming: Computation issues for branch-and-cut algorithms // Computational Combinatorial Optimization / M. Jünger, D. Naddef (eds). Springer Verlag, 2001. p. 1-25.

114. Modern heuristic techniques for combinatorial problems / C. R. Reeves (ed.). London: McGraw-Hill. 1995. 320 p.

115. Martí R., Reinelt G., Duarte A. A Benchmark Library and a Comparison of Heuristic Methods for the Linear Ordering Problem. Computational Optimization and Applications 51(3), 1297-1317 (2012). doi:10.1007/s10589-010-9384-9

116. Martí R., Reinelt G. The Linear Ordering Problem. Exact and Heuristic Methods in Combinatorial Optimization. Springer, Heidelberg (2011).

117. Nemhauser G. L., Savelsbergh M. W. P., and Sigismondi G. S. MINTO, a Mixed INTeger Optimizer // Operations Research Letters. 1994. V. 15. P. 47-58.

118. Nemhauser G. L., Wolsey L. A. Integer and combinatorial programming. Chichester: John Wiley & Sons. 1988. 763 p.

119. Osman I. H., Laporte G. Metaheuristics: A bibliography. Ann. Operat. Res. 1996. V. 63. P. 513-628.99. Ouaarab, A., Ahiod, B., Yang, X.S.: Discrete cuckoo search algorithm for the travelling salesman problem. Neural Computing and Applications 24(7-8) (2014) 1659–1669

120. Papadimitriou C. H. On the complexity of integer programming. Journal of the ACM. 1981. V. 28. P. 765-768.

121. Parker R. G., Rardin R. L. An overview of complexity theory in discrete optimization: Part 1. Concepts // HE Transactions. 1982. V. 14. P. 3-10.

122. Parker R. G., Rardin R. L. An overview of complexity in discrete optimization: part II. Results and implications // HE Transactions. 1982. V. 14. P. 83-89.

123. Ralphs T. K., Galati M. Decomposition in Integer Programming. Integer Programming: Theory and Practice / J. Karlof, ed. 2005. P. 57.

124. Reinelt, G. Tsplib: A traveling salesman problem library. ORSA journal on computing 3(4) (1991) 376–384.

125. Reinelt, G. Fast Heuristics for Large Geometric Traveling Salesman Problems. ORSA Journal on Computing 4(2), 206-217 (1992). doi: 10.1287/ijoc.4.2.206

126. Reinelt G. TSPLIB - A Traveling Salesman Problem Library. ORSA Journal on Computing 3, 376-384 (1991).

127. Rebennack S., Reinelt G., Pardalos P.M. A tutorial on branch and cut algorithms for the maximum stable set problem. International Transactions in Operational Research 19(1-2), 161-199 (2012). doi:10.1111/j.1475-3995.2011.00805.x

128. Rebennack S., Oswald M., Theis D.O., Seitz H., Reinelt G., Pardalos P.M.: A Branch and Cut solver for the maximum stable set problem. Journal of Combinatorial Optimization 21(4), 434-457 (2011). doi:10.1007/s10878-009-9264-3

129. Sager S. Numerical methods for mixed integer optimal control problems. Dissertation, Universität Heidelberg (2006).

130. Sergienko I. V., Shylo V. P., Roshchyn V. O. Algorithm unions for solving discrete optimization problems. Cybern. Syst. Anal. 59, No. 5, 753-762 (2023).

131. Skakalina O., Kapiton A. Comparative analysis the application of heuristic algorithms for solving the TSP problem. Control, Navigation and Communication Systems, no. 2, 2024, pp. 144-151.

132. Speth, M.: Exact Solutions for Discrete Grahical Models: Multicuts and reduction Techniques. Dissertation, Universität Heidelberg (2014).

133. Sweeney D., Murphy R. A method of decomposition for integer programs // Operations Research. 1979. V. 27, N 6. P. 1128-1141.

134. Syslo M. M., Deo N., Kowalik J.S. Discrete Optimization Algorithms with Pascal Programs. Englewood Cliffs: Prentice-Hall, 1983. 542 p.

135. Tarjan R.E. Complexity of combinatorial algorithms // SIAM Review. 1978. V. 20. P. 457-492.

136. Tawarmalani M., Sahinidis N. V. Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications. Kluwer Academic Publishers. 2002. 475 p.

137. Terzi D. On heuristics and test construction for the experimental study of the traveling salesman problem. Scientific Collection ”InterConf”. Vol. 3(39). 2020. Manchester, Great Britain, ISBN 978-0-216-01072-7.

138. Taha H.A. Integer prigramming: theory, applications and computations. NewYork-London, Acad.Press., 1975, 388 p.

139. Terzi D. Economic forecasting using neural networs, Ediţia a IX a Conferinţei ştiinţifice internaţionale “Polotici şi mecanisme de inovare şi dezvoltare a proceselor economico-financiare şi sociale în plan naţional şi internaţional”. Chişinău, 2014. p. 381-385.

140. Terzi D. Search Algorithms for a Sistem of Different Reprezentatives of Subsets of a Finite Set. Journal of Economics and Management Sciences, Vol.6, No 2, 2023, p..11-18. https://doi.org/10.30560/jems.v6n2p11

141. Terzi D. A natural approach to solving the traveling salesman problem. Кібернетика та комп'ютерні технології. Nr. 4(15), 2023, p.43-52. https://doi.org/10.34229/2707-451X.23.4.6 http://cctech.org.ua/ua/vypask-2023-4-mnu

142. Terzi D. Algorithm for constructing the Traveling Salesman Problem with a Given Optimal Solution. Intellectus, 1/2023, p.171-178. DOI: https://doi.org/10.56329/1810-7087.23.1.17

143. Terzi D. On the procedure for constructing ortogonal cyclic permutations and its application. Conferința științifico-practică internațională "Știința, Educație. Cultura", A 32-a aniversare a USC, Culegere de Articole, Vol. 1 Secțiunea III Tehnologii informaționale, matematica, fizica, Comrat, 2023, p.457-461. https://orcid.org/0000-0003-5489-5058

144. Terzi D. Methods for constructing system of different representatives and their application. Intellectus, 2/2023, p. 131-137. https://doi.org/10.56329/1810-7087.23.2.14

145. Terzi D. On one implementation of a natural approach to solving the traveling salesman problem. Cybernetics and Computer technologies, No. 2(18), 2024, p. 31-38. DOI: 10.34229/2707-451X.24.2.3.

146. Terzi D. Optimality criterion and algorithm for solving the Traveling Salesman Problem. Proceedings IMCS-60. Intern Conference dedicated to the 60th Anniversary of the foundation of Vladimir Andrunachievici Institute of Mathematics and Computer Science, Moldova State University, October, 10-13, 2024. Chisinau, p. 476-482. Republica Moldova Agenția de Stat pentru Proprietatea Intelectuală. IDNP: 0980912012015553. Data înregistrării: 04.04.2024.

147. Traveling Salesman Problem: a Prospective Review of Recent Research and New Results with Bio-Inspired Metaheuristics and Novelty Search Eneko Osaba, Xin-She Yang, Javier Del Se References.

148. Taşkesen Bahar, Shafieezadeh-Abadeh Soroosh, Kuhn Daniel. Semi-discrete optimal transport: hardness, regularization and numerical solution. Math. Program. 199, No. 1-2 (A), 1033-1106 (2023).

149. Vasyanin V. A., Trofymchuk O. M., Ushakova L. P. A methodology of the mathematical modeling for perspective development of nodes and transport routes in the multicommodity hierarchical network. I: Optimization problems. Cybern. Syst. Anal. 60, No. 1, 72-83 (2024); translation from Kibern. Sist. Anal. 60, No. 1, 87-101 (2024).

150. Vo T.K. Exact and Heuristic Solutions to the Bandwidth Minimization Problem. Dissertation, Universität Heidelberg (2011).

151. Wang P., Reinelt G., Gao P., Tan Y. A model, a heuristic and a decision support system to solve the scheduling problem of an earth observing satellite constellation. Computers and Industrial Engineering 61(2), 322-335 (2011). doi:10.1016/j.cie.2011.02.015.

152. Wedyan A., Whalley J., Narayanan A. Solving the traveling salesman problem using hydrological cycle algorithm. American Journal of Operations Research 8(03) (2018).

153. Wedelin D. An algorithm for large scale 0-1 integer optimization with application to airline crew scheduling. Annals of Operations Research. 1995. v. 57. P. 283-301.

154. Woeginger G. J. Exact algorithms for NP-hard problems: A survey "Combinatorial Optimization Eureka! You shrink!" / M. Juenger, G. Reinelt and G. Rinaldi (eds.). LNCS 2570. Springer. 2003. P. 185-207.

155. Yagiura M., Ibaraki T. On metaheuristic algorithms for combinatorial optimization problems. Systems and Computers in Japan. 2001. V. 32. P. 33-55.

Dmitri TERZI

NUMERICAL METHODS OF DISCRETE OPTIMIZATION

Bun de tipar 30.12.2024.
Formatul 70×100 1/12 Coli de tipar 15,8.
Coli editoriale 7,40. Comanda 127.
Tirajul 50 ex.

Centrul Editorial-Poligrafic al USM
str. Al. Mateevici, 60, Chişinău, MD-2009
e-mail: cep1usm@mail.ru; usmcep@mail.ru