# Software Implementation of Genetic Algorithm for Optimization of Cargo Placement in the Conditions of limited Warehouse Space

NATALIA MAMEDOVA[1], YULIA KHIZHNYAKOVA[2]
[1]Basic Department of Digital Economy,
Plekhanov Russian University of Economics,
36, Stremyanny Lane, Moscow, 117997,
RUSSIA

[2]Development Department,
LAS LLC,
14k1, Murmansk passage, Moscow, 129075,
RUSSIA

*Abstract:* - In this paper, we propose a software implementation to solve the mathematical problem of optimal placement of cargo units on the territory of a multimodal transport and logistics center. Warehouse management in intermodal and multimodal transportation is complicated by the problem of selecting an assortment of cargo in conditions of limited storage space. The solution to this problem should be mathematically correct, automatizable, and scalable, since different types of warehouses and different transport systems are concentrated in the territory of multimodal transport and logistics centers. We propose to apply the genetic algorithm as a mathematical apparatus for solving the above problem and a ready-made software implementation for the optimal placement of cargo units. The algorithm determines the optimal subset of cargo units that can be placed in the warehouse taking into account the weight and value priority constraints of the selected cargo units.

*Key-Words:* - Programming, genetic algorithm, warehouse management system, multimodal transport and logistics center, functional requirements.

Received: May 14, 2024. Revised: December 11, 2024. Accepted: January 16, 2025. Published: April 2, 2025.

## 1 Introduction

Combinatorial optimization is a field of mathematical optimization that deals with the search for the best combinations of elements from a given set to achieve a certain goal. From the mathematical point of view, the search includes maximization or minimization of some function associated with these elements. In logistics as well as in other fields combinatorial optimization is applied in goal setting, planning, and resource allocation. Finding optimal solutions under constraints makes this area of mathematical optimization an important tool for improving the efficiency and productivity of logistics operations.

A traditional problem for the formulation of the combinatorial optimization problem is the determination of the most efficient routes for cargo delivery, including for several directions. In particular, solutions for routing dependent or independent vehicles are proposed, considering the limits of their capacity and the length of the route with some number of locations for stops [1], as well as taking into account the state of traffic in real-time, [2]. The complexity of solving such problems is increased by introducing additional parameters, such as heterogeneity of the fleet of vehicles and warehouses, and the presence of time windows in the process of picking and delivery of cargo, [3], [4].

Popular for implementation are solutions for finding the time-optimal route of one or several agents between objects, [5], and the problem can be complicated by the associated operations of delivery, placement of cargo in the warehouse, packing, packaging with changing the format, weight, and size of the cargo unit, [6]. Of high practical significance are the solutions to the multi transportation problem of a traveling customer who, in addition to determining the optimal route, faces the problem of planning purchases at markets, [7].

Common in these solutions is the search for the optimum with the lowest system cost. But since the

optimal cost of the combinatorial solution should tend to zero, its obtaining turns out to be eventually impossible under the existing constraints. Therefore, the search for the optimum involves such additional aspects as reducing the complexity of the solution process, speed of finding a solution, compliance with constraints and practical applicability.

The factors of intermodality and multimodality of transportation seriously complicate the problem of combinatorial optimization. At the same time, less attention is paid to the problems of optimization of warehouse space management in transport and logistics centers than, for example, to the search for optimal selection of transport modes involved in intermodal or multimodal transportation. When designing for several modes of transport a network of circulation service of the territory, [8] or in network optimization models for determining the order and placement of materials at production facilities, [9] the focus of attention is on routing - the solution should provide maximum throughput, minimize waiting time and storage costs.

It is not possible to use off-the-shelf combinatorial optimization solutions for transport routing in warehouse management systems. Off-the-shelf solutions are oriented to agents engaged in transportation or ordering transportation, but not to warehouse logistics agents. Warehouse logistics has its own specifics. There is a fleet of material handling equipment - cranes, industrial forklifts, conveyors, conveyors, and others - all in a single mutually determined complex relationship. There is the warehouse space, which can vary in warehouse type, capacity, specifics of placement, movement, and storage of cargo units.

To present the scope of the specific features of the warehouse management system, we give a classification of one of the types of warehouse - a tiered warehouse is subdivided into the following subtypes:

- high-bay warehouse;
- ceiling rack warehouse;
- pallet racking warehouse;
- cantilever racking warehouse;
- cellular warehouse;
- through-racked warehouse;
- drive-through racking warehouse;
- a warehouse with circulating racks.

On the territory of a multimodal transport and logistics center there may be several types of warehouses or several sub-types of warehouse. And their management from the point of view of a complex system requires multi-criteria and complex solutions. This becomes obvious if we take into account that a multimodal transport and logistics center is located in a network-wide (multimodal) transport hub and serves several modes of transport when combining the technology of cargo processing at the terminals included in its composition, [10].

In comparison with intermodal transportation, the multimodal type of transportation is complicated precisely by the level of responsibility of the operator, who controls all stages of warehousing and internal movement. The operator of a multimodal transport and logistics center provides a full cycle of basic and end-to-end physical processes in the warehouse, and operations related to the use of different modes of transport. For example, cargo arrives by rail to the warehouse (in bulk), then it is processed, reloaded, and packed (transshipment), followed by loading on a sea vessel and shipment of cargo.

Specialized warehouses for cargo storage and processing in the architecture of a multimodal transport and logistics center are independent clusters united in a single network topology. For the effective functioning of the entire network of warehouses, it is necessary that the work of each cluster is organized optimally.

Thus, there is a space for the action of optimization models and there is a need for solutions to combinatorial optimization problems. We see the main task as the development of a solution that optimizes the management of warehouse space within a separate cluster in the territory of a multimodal transport and logistics center. The complexity of the problem is increased by taking into account the parameters characterizing the cargo units placed in the warehouse.

If we qualify the problem in mathematical terms, we are dealing with an NP-complete problem. We assume that for a task from the class NP, it is possible to reduce any other task from this class to it in polynomial time. This means that if a polynomial algorithm is found for an NP-complete problem, then all problems from the NP class can also be solved in polynomial time. Thus, the complexity bounds of the sought algorithm are defined.

The combinatorial optimization algorithms associated with the problem are NP-complete in nature, which makes it appropriate to investigate ways to speed up and simplify the procedures for solving these problems. The proposed research includes material describing and justifying the application of an approximate combinatorial optimization algorithm, the genetic algorithm, to solve the problem at hand. The software implementation of the application of genetic algorithm for optimization of warehouse space management is developed. The presented research

results effectively fulfill the requirements to reduce the complexity of the solution process, take into account the constraints to avoid the algorithm hitting the local optimum, and have practical significance. The performance is experimentally evaluated using machine experiment results. The posed problem can be transformed by introducing new parameters or horizontal scaling to additional clusters of a multimodal transport and logistics center.

## 2 Approaches and Algorithms for Optimization of Warehouse Processes

The need to optimize various aspects of the warehouse management system is great. The subject area of optimization can be physical processes and warehouse operations in aggregate or individually. However, the local optimum found for a warehouse operation or even a process is limited to a local search. And this, on the one hand, does not allow us to be sufficiently confident in the convergence of the solution, and on the other hand, the result based on the local search can not be extended to global warehouse processes. Hence the a need to design and implement solutions based on globally optimal algorithms.

Methods that generate a population consisting of samples of points in parameter space, perform a parallel evaluation of their proximity to a target, and then recombine them in some way to bring the population to a global optimum have been known for a long time and have not lost their relevance, [11], [12], [13]. An overview of the methods used to achieve local or global optima is presented with more objectivity in academic publications, [14], [15], [16] and abstract studies, [17], [18], [19], [20]. However, selecting and justifying the application of mathematical apparatus and methods to solve the problem at hand is a necessary step in conducting the study. For this purpose, it was necessary to define both the subject area of the research and objectify its choice taking into account the implementation requirements (including software implementation) and limitations associated with the initial data. Otherwise, the result of determining the global optimum by one or several methods will be unavailable for machine experimentation.

For this study the subject area was chosen as the problems of optimization of cargo placement in the warehouses of a multimodal transport and logistics center.

Limited storage space creates variability in the placement of cargo units, so the prioritization should be based on some parameters. In the conditions of specificity of intermodal and multimodal transportation such parameter is the value of cargo unit, and the integer value can be taken into account as a qualitative or quantitative attribute. Another parameter is the weight of a cargo unit, which is a classical approach to the organization of warehousing, and the parameter is accepted by us from the consideration that on the territory of a multimodal transport and logistics center, there may be warehouses of different types, in the management of which there is something in common.

If we simplify as much as possible the combinatorial optimization problem about the placement of cargo units in the warehouse, we obtain the classical “knapsack problem” - it is necessary to put a certain number of items in it, having at our disposal data on the parameters of knapsack and the items themselves, [21].

Despite the different properties of the problem and its wide application in various fields ranging from linguistics to cryptography, it is formally solved by the method of enumeration, [22]. Hypothetically, each item can end up in a knapsack, and therefore the selection of a set of items is done by taking into account certain inputs. In our case, these are the value and weight parameters of the cargo units.

The Knapsack Problem has several varieties, each with different conditions and constraints:

1. 0-1 Knapsack Problem. This is the basic form of the Knapsack Problem in which: each item can be either included in the knapsack (value 1) or excluded (value 0). The goal is to maximize the total value of the selected items, provided that their total weight does not exceed a given knapsack capacity.
2. Bounded Knapsack Problem. In this variation, each item can be selected a limited number of times (e.g., no more than 3). This allows for cases where items are available in limited quantities.
3. Unbounded Knapsack Problem. In this version, each item can be selected an unlimited number of times. This is suitable for situations where items can be reused or ordered in large quantities.
4. Multi-dimensional Knapsack Problem. This problem includes several constraints (e.g., weight and volume), which makes it more complex. Each item has multiple weights corresponding to different constraints. The goal is to maximize the total value while satisfying all constraints.
5. Multiple Knapsack Problem. Here there are

multiple knapsacks, each with a different capacity. The problem is to distribute items among multiple knapsacks to maximize the total value.

6. Multiple-Choice Knapsack Problem. In this variation, items are divided into groups, and you must choose exactly one item from each group. This adds an extra level of difficulty to the problem.

Of all the varieties of problem statements, we chose the one that best lends itself to further transformation of the conditions imposed on the warehouse space - the "0-1 Knapsack Problem". Besides, it has a clear structure and conditions, which makes it a universal tool, as we do not limit the possibility of implementing the obtained solution in any of the MTLC warehouses. But, if we take into account the mechanism of pre-sorting before placement in one of the MTLC warehouses, then the mathematical apparatus of the solution should be based on the "Multiple Knapsack Problems" variety. The variety "Multi-dimensional Knapsack Problem" is redundant for the current formulation of the problem, since the parameter "value" already has a complex character and includes the cost of placement in the territory of the warehouse.

Accordingly, the basic condition is defined as follows: a cargo unit can be either taken in its entirety or not taken at all.

In warehouse logistics, goods can also have different parameters such as turnover, shelf life, and storage condition requirements. This information can be integrated into the model of the knapsack problem, allowing us to take into account not only physical constraints but also economic aspects of goods storage and handling.

The resulting solution can be further complicated by selecting the "Multi-dimensional knapsack problem" condition, adding the "loading time" parameter, or the "Multiple Knapsack Problem" condition, if it is necessary to solve the problem for several adjacent warehouses within a multimodal transport and logistics center.

The brute-force method is used in a number of algorithms that can be compared to each other. For example, the brute-force method and the dynamic programming method are both exact algorithms whose operation leads to an exact solution. However, while the complexity of the brute-force method is O(n!), the complexity of the dynamic programming method is O(w*n). Despite the simplicity of implementation and independence from the type of input data, the brute-force method is time-consuming. In addition, we assume that the number of possible solutions to the problem is large, so the brute-force method is not applicable to this study. The method of dynamic programming has the same advantages, but there is no universal algorithm for its application, and obtaining an accurate result requires a large amount of computational work. The method of branches and bounds by analogy with the method of complete enumeration has the same mechanism of work and can reduce time costs, but it is very sensitive to the initial data.

Among the algorithms that do not provide a single correct solution but offer approximate accuracy are the greedy algorithm and the genetic algorithm. Greedy algorithms take locally optimal solutions at each step, hoping that the final solution will also be optimal. However, they do not always lead to a globally optimal solution, especially in complex problems such as the traveling salesman problem or the graph partitioning problem. Genetic algorithms are stochastic methods that utilize the mechanisms of natural selection and mutation to find solutions. They can find good approximate solutions in complex search spaces, but do not guarantee to find the optimal solution.

The greedy algorithm, having complexity O(n*log(n)), is simple enough in realization and can work with large values of n, but its essential disadvantage for application in the chosen subject area is that the convergence of the solution is conditioned mainly by the value parameter of the cargo unit, and the warehouse parameters are derived in calculations. In the long run, this may lead to inefficient solutions to the problem at hand.

Thus, if in the structure of requirements for the choice of algorithm a combination of solution speed, insensitivity to the initial data, the ability to work with several factors, and average computational power is stated, then the optimal option is the choice of genetic algorithm. The genetic algorithm has high speed, can handle large values of n, is independent of the type of input data, and does not require significant computational power. And, although the algorithm does not guarantee finding the only correct solution, it always finds the best result among the possible ones, [23].

The genetic algorithm uses a stochastic approach, which allows it to find good solutions in a reasonable amount of time even when the size of the input data increases significantly. The genetic algorithm works based on a population of possible solutions, which allows it to explore the solution space more efficiently. Unlike deterministic methods that can get stuck in local minima, genetic algorithms use selection, crossover, and mutation mechanisms to create new solutions. This helps avoid premature convergence and contributes to

finding better solutions. A genetic algorithm can easily adapt to different problem conditions. The user can change parameters such as population size, mutation probability, and selection methods, which allows optimizing the search process depending on the specific requirements of the problem. All these aspects and the results of comparison with other methods are the basis for choosing a genetic algorithm as a tool for solving the problem.

# 3 The Nature and Stages of a Genetic Algorithm

The genetic algorithm belongs to a group of evolutionary computation methods that combine various applications of evolutionary principles. John Holland proposed the concept of encoding solutions in the form of "chromosomes", applying crossing and mutation operators to create new solutions, and evaluating and selecting these solutions based on their relevance. He explored their potential for solving complex optimization problems and developing adaptive systems, [24].

The operation of the algorithm consists of searching for elite instances in the population, and the evolutionary process ends with finding a satisfactory solution. Of course, many evolutionary simulations with increasing instances in the population will require an increase in computational resources. But in the case of the problem we are solving, we operate not with individual cargo units, but with groups of them formed according to the chosen value and weight parameters.

The main stage of genetic algorithm operation is a cyclic sequence.

Population initialization. The genetic algorithm starts by creating an initial population consisting of randomly generated individuals. Each individual is represented as a chromosome that encodes a potential solution to the problem (1 for a usable cargo unit and 0 for a non-usable one). A solution in the population is a set of such chromosomes - a genome.

Evaluation of adaptability. The fitness function is usually determined on a task-specific basis - all necessary requirements are considered to find the "optimality" of the solution. Each individual in a population is assessed by an adaptation function, which measures how well an individual solves a problem. This determines which individuals are more "successful" than others and therefore have a better chance of survival and reproduction. The higher the value of the fitness function, the better the solution. This allows the best-adapted solutions to be selected to pass on their characteristics to the next generation of the population.

Once the population is evaluated, individuals are selected for crossover to create a new generation. The genetic algorithm allows the use of different selection methods (e.g., roulette or tournament selection) and different crossover strategies (single, double, etc.), allowing you to experiment with approaches to achieve the best results.

Selection. Based on the values of the fitness function, individuals are selected to create the next generation. There are various selection methods such as rank selection, and tournament selection (roulette wheel selection), that help to retain the best-adapted individuals while maintaining the diversity of the population.

Crossbreeding (crossover). Selected individuals are crossed to create offspring. This process involves combining the genetic information of the two parents to create new individuals with characteristics of both parents. Crossover occurs by combining the genetic components of the parental individuals, resulting in offspring with new combinations. There are several methods of crossover, including single-point and multi-point crossover. This allows you to introduce diversity into a population and explore different combinations of genetic components to find the best solutions for a particular problem.

Mutation. To allow new solutions to appear in the population (increase genetic diversity) and prevent premature convergence of the algorithm, a mutation operation is used. This is a random change in the genetic components of an individual (one or more genes in a chromosome) that can lead to new solution variants - potentially more successful solutions. This helps to explore a wider search space.

To tune the mutation parameters in a genetic algorithm used to optimize the placement of cargo units in a warehouse, it is important to consider several key aspects. The optimal mutation percentage typically ranges from 1% to 10% of the total number of genes in a chromosome. A low percentage can lead to a lack of diversity, while too high can disrupt useful combinations of placement in limited warehouse space.

Different types of mutational operations can be applied: a) random exchange: the exchange of two randomly selected genes; b) random permutation: random rearrangement of the gene sequence; c) inversion: inverting the order of a particular segment of the chromosome. The inversion type is

used to solve the problem at hand because this operation introduces the least entropy into the research process. After crossing and mutation, a new generation is created, which replaces the previous generation.

Population replacement. The new individuals created by crossover and mutation replace the old population, and the process repeats. This cycle continues until an optimal or near-optimal outcome is achieved (depending on the stipulated stopping condition). Examples of a stopping condition include, but are not limited to, a maximum number of generations, the achievement of a given level of adaptability, or the achievement of a certain level of stability of strong solutions.

The operators of the genetic algorithm work together to provide an efficient search for optimal solutions through a combination of selecting the best individuals, creating new offspring through interbreeding, and maintaining diversity through mutation.

The benefits and experience in implementing the results of optimization problem- solving drew attention to the genetic algorithm, expanding its field of application. Since then, the number of publications and interest in this field has increased significantly, opening new perspectives with the application of artificial intelligence, [25]. However, a genetic algorithm is not a panacea for solving all optimization problems, like any algorithm, it should be applied consciously and reasonably.

# 4 Description of the Logic of Genetic Algorithm Application

Genetic algorithms can get stuck in a local optimum for reasons related to their mechanism of operation and population structure. The influence of factors contributing to this phenomenon should be avoided, since their manifestation may distort or reduce the efficiency of genetic algorithm application.

A lack of diversity in the population under study should be avoided. A population that is close to homogeneity will cause the algorithm to explore a limited solution space. As a result, individuals may get stuck in local optima because there are no alternative paths to explore. In the current study, this factor is offset by the diversity of cargo unit types that can potentially fill the storage space.

Strong selection, in turn, may result in the algorithm being unable to explore a wider range of solutions, getting stuck in a local optimum. In the practical domain, if individuals are selected too aggressively based on their fitness, there is a risk that less fit but potentially more promising solutions will be weeded out. This factor is considered a moderate risk, which is managed by dynamically tracking and changing the value parameter of the cargo unit when necessary.

A small number of mutations is a factor that limits the possibility of finding new solutions. The frequency with which a mutation is applied during evolution is also important. If the mutation probability is low, new genetic combinations appear rarely, which can also contribute to getting stuck in local optima. This factor qualifies as a significant risk in the current study, and its management has two trajectories - risk acceptance and splitting the group of cargo units into smaller batches. The choice of trajectory is determined by the value parameter - the group of cargo units with higher value relative to the groups in the random population is split into parts in the proportion of 2:3:5, and the group with lower value remains within the boundaries of the natural mutation process. This approach allows the algorithm to utilize computational resources efficiently. Groups with different adaptability can generate a variety of solutions, which increases the chances of finding an optimal solution without significant computation time.

Thus, groups with higher value (50%) are labeled as highly adapted, this preserves diversity but prevents premature convergence to local minima. Theoretically, the application of a 1:1:1:1 proportion provides equal access to selection for all individuals, regardless of their level of adaptation, but the choice of this proportion will not allow for differences in cargo value.

In addition, it was decided to apply mutation only in every second or third iteration in order to maintain population stability and avoid excessive variability.

The structure of the target function is also a limiting factor for the genetic algorithm. If the target function has many local optima, the genetic algorithm may get stuck in one of them. The influence of this factor is reduced by entering into the work of the research mechanism those data that were obtained earlier for the warehouse space under study, or the data of comparable warehouse space on the territory of a multimodal transport and logistics center.

Problems with crossover refer to vulnerabilities in the very workings of the research mechanism. If the logic of crossover implementation does not ensure the mixing of genetic information, it leads to the fact that new individuals of a random population will be too similar to their parents and will not be able to explore new solution areas. In this study, the

potential problems of crossover performance are addressed by preserving the replacement history of the population and introducing crossover re-execution functions if the similarity between generations is above a limit value of 0.75.

A value of 0.75 represents a compromise between the speed of convergence and exploration of the solution space. Too low a convergence value can slow down the process because new generations will be significantly different from previous generations, which can lead to the loss of good solutions already found. On the other hand, a value of 0.75 preserves a sufficient number of good genes from previous generations, which contributes to faster finding of optimal solutions. Similarity within 0.75 allows the efficient use of selection and crossover methods and indicates that the progeny of the new generation should be sufficiently different from their ancestors. This avoids premature convergence to local minima and helps to maintain diversity in the population. If the similarity between generations is too high (e.g., 0.9 or higher), it may cause the algorithm to get stuck in the same solution without exploring other possible options.

The described factors of algorithm closure in local optimums and solutions to contain their influence emphasize the importance of maintaining diversity in the population, proper choice of selection and mutation operators, and consideration of the structure of the target function when using genetic algorithms to minimize risks and manage risks efficiently.

# 5 Building a Mathematical Model

Within the warehouse operation optimization problem, a genetic algorithm is used to minimize the cost of allocating cargo units in a constrained warehouse space. Each chromosome represents a sequence of warehouse operation execution for allocation. The process of solving the problem of placing cargo units in the warehouse space taking into account the given cargo parameters on the basis of a genetic algorithm consists of several steps:

1. Generation of the initial population. Randomly generate an initial population of sets of cargo units, where each set represents a possible solution (combination of cargo units to be placed in the warehouse space).
2. Adaptability Assessment. Each set of cargo units is assigned an adaptability value according to a target function - the total value of the cargo units in the set.
3. Selection. From the initial population, “parents” are randomly selected to create a new population based on their fitness values.4. Crossbreeding. The selected parents are combined to create new offspring. A crossover operator will be used, where breakpoints in the sets of parents are randomly selected and the offspring inherit parts of items from each parent.
5. Mutation. A random chromosome representing the order of the cargo units is selected from the current population. Next, the order of a particular segment of the chromosome is inverted. Sequentially, a mutation occurs, making random changes to the set of cargo units - randomly adding or removing cargo units from the set. After the mutation step, the new chromosome can lead to a change in fitness (placement cost), which forms the best solution to the optimization problem.
6. Reproduction. A new population is formed by combining selected, crossed, and possibly mutated sets of cargo units.
7. Evaluation of the new population. The cycle procedures, from evaluation of fitness to reproduction, are repeated until the stopping criterion is reached. Stopping will occur when a sufficient level of stability of strong solutions is achieved.
8. Completion. The expected outcome of the genetic algorithm is the optimal set of cargo units from the last population. This set will represent the optimal solution to the warehouse space allocation problem, given the value and weight constraints of the cargo units.

To solve the problem of optimization of warehouse operations on the basis of a genetic algorithm, the mathematical apparatus with the problem formulation “0-1 Knapsack Problem” was used. This formulation is an important example of an optimization problem with constraints. The problem illustrates the basic principles of resource selection and allocation in the presence of constraints and is the basis for the development of various algorithms for solving such problems. It consists of selecting a set of items with given weights and values so as to maximize the total value of the items placed in a backpack without exceeding its maximum capacity. Understanding this problem helps in further studying more complex optimization problems and developing efficient algorithms to solve them.

There is a warehouse with bounded space $C$ and a set of $n$ cargo units, each of which has a certain weight $w_i$ and value $v_i$. It is necessary to choose such cargo units that their total weight does not exceed the volume of the warehouse space, and the total value is maximized.

Let: $n$ be the number of cargo units, $w_i$ be the weight of cargo unit $i$, $v_i$ be the value of cargo unit $i$, $C$ be the maximum warehouse space, $x_i$ be a variable that takes the value 1 if cargo unit $i$ is placed in the warehouse and 0 otherwise.

Given the "0-1 Knapsack Problem" condition, the problem is formulated as follows: maximize $max \sum_{i=1}^{n} v_i x_i$ under the constraints:

$$\sum_{i=1}^{n} v_i x_i \leq C,\ x_i \in \{0, 1\},$$
$$i = 1, \ldots, n.$$

This constraint ensures that the total weight of the selected items does not exceed the capacity of the backpack.

In this formulation, the problem is represented as follows:

Target function: $\sum_{i=1}^{n} v_i x_i$

The constraint $\sum_{i=1}^{n} v_i x_i \leq C$ reflects the physical constraints and guarantees that the total weight of the selected items does not exceed the capacity of the backpack.

Binary variables: $x_i \in \{0, 1\}$, meaning that each item can either be included in the backpack ($x_i = 1$) or excluded ($x_i = 0$). This makes the problem discrete and allows combinatorial optimization methods to be used to solve it.

## 6 Applying a Genetic Algorithm to Solve a Problem

In this section, a genetic algorithm will be implemented and a program will be written to run an application to find an optimal solution to the placement of cargo units in a limited warehouse space. The workflow of the software duplicates the steps of the genetic algorithm.

The BackpackItem class represents a cargo unit in the optimal warehouse occupancy problem. The following fields of this class are highlighted:

- Name: a string field designed to store the name of the cargo item.
- Weight: an integer field representing the weight of the cargo unit.
- Worth: An integer field representing the value or cost of the cargo unit.

```
internal class BackpackItem
{
    /// <summary>
    /// Name, weight and value.
    /// </summary>
    public string Name;
    public int Weight;
    public int Worth;

    /// <summary>
    /// Class constructor that initializes object parameters.
    /// </summary>
    /// <param name="weigth"></param>
    /// <param name="worth"></param>
    /// <param name="name"></param>
    public BackpackItem(int weigth, int worth, string name)
    {
        Weight = weigth;
        Worth = worth;
        Name = name;
    }
```

The BackpackGenome class describes a genome and represents a separate warehouse on the territory of a multimodal transport and logistics center for the work of the algorithm. If we refer to the terminology of genetic algorithm, the warehouse can be represented as a genome. Its description is represented by the following fields:

- Fitness: a real field designed to store the fitness value of the genome. Fitness reflects the degree of suitability of an individual warehouse to accommodate and store the selected cargo item.
- ItemsPicked: a list of objects of class BackpackItem representing cargo units selected for placement in this warehouse (in genetic algorithm terminology - for packing in this genome). This list of objects stores information about which cargo units were included in this optimal placement solution.
- Parameter: an integer field representing the warehouse parameter as a number obtained by converting a string of string from a binary value to a decimal value.

```
internal class BackpackGenome
{
    /// <summary>
    /// The importance of genome adaptation and a list of selected subjects.
    /// </summary>
    public float Fitness;
    public List<BackpackItem> ItemsPicked = new();
    /// <summary>
    /// Genome parameter.
    /// </summary>
    public int Parameter;
    /// <summary>
    /// Class constructor that initializes genome parameters.
    /// </summary>
    /// <param name="parameter"></param>
    public BackpackGenome(int parameter)
    {
        Parameter = parameter;
        Fitness = 0;
    }
```

The BackpackSolver class represents the genetic algorithm itself. The methods and variables used will be described below.

- CrossOverProbability, MutationProbability, PopulationSize, GenerationCount - constants representing crossing and mutation probabilities, population size and number of generations.
- MaxValue - genome capacity.
- crossPoint, mutatePoint, child1, child2 - auxiliary variables for crossing and mutation.
- NextGeneration and Solutions - lists of objects of BackpackGenome class to represent the next generation and current solutions.
- bestFitness - a field to store the best genome with the best fitness.
- crossedGenomes and crossoverPartner - fields for storing crossed genomes and crossing partner results.
- RunGeneticAlgorithm - method to run the genetic algorithm. Outputs results to the console, including selected items, maximum weight, current weight, and current value.
- Evolve - Method for evolving the genetic algorithm, including generating the next generation, crossbreeding, mutating, and selecting the best genomes.
- CalculateFitness - A method for calculating genome fitness given the weight and value of items.
- Crossover and Mutation - methods for performing genome crossover and mutation operations.
- GenerateRandomSolutions - method for generating random genomes for an initial population.

## 7 Generation of the Initial Population, Generate Random Solutions

When the user enters specific integer values into the fields of the BackpackItem class, the algorithm creates an initial population of random solutions.

- populationSize - method parameter indicating how many genomes should be generated for the initial population.
- temp - temporary list of genomes to be returned by the method.
- Cycle for (i from 0 to populationSize - 1) to generate the required number of genomes.
- new BackpackGenome(Rnd.Next(1, Int32.MaxValue)) - creation of a new genome using the constructor of BackpackGenome class. A random integer from 1 to Int32.MaxValue is passed to the constructor parameter.

```
public static List<BackpackGenome> GenerateRandomSolutions(int populationSize)
{
    // Temporary list for storing generated genomes.
    var temp = new List<BackpackGenome>();
    // Each item in the backpack is represented by one bit, the maximum value of the parameter is the number that has all bits set to 1.
    int maxParameter = (1 << Selection.Count) - 1;
    // Maximum number of attempts.
    int maxAttempts = 1000;
    for (var i = 0; i < populationSize; i++)
    {
        // Initialization, genome parameter = 0.
        int randomParameter = 0;
        // Remaining weight, initially = 0.
        int remainingWeight = MaxValue;
        // Attempt number.
        int attempts = 0;
```

## 8 Adaptability Function, Calculate Fitness

The implementation of the genome fitness function by the CalculateFitness method is done as follows:

- Summarizing the value of the selected items in the Fitness property of the genome.
- Calculating the penalty for exceeding the maximum weight - the penalty is calculated based on the difference between the total weight of the selected items and the maximum payload value to limit decisions that lead to overloading.
- Once the accommodation is calculated, a check is performed. If the fitness value is found to be less than 1, it is corrected and set to 1. This ensures that the genome's fitness is not negative.

```
public static void CalculateFitness(BackpackGenome genom)
{
    // Summarizing the value of selected items.
    genom.Fitness += genom.ItemsPicked.Sum(t => t.Worth);
}
/// <summary>
/// Genetic crossbreeding.
/// </summary>
/// <param name="parent1"> Parent 1.</param>
/// <param name="parent2"> Parent 2.</param>
/// <returns></returns>
public static int[] Crossover(BackpackGenome parent1, BackpackGenome parent2)
{
    _crossPoint = Rnd.Next(1, Selection.Count - 1);
    // Create a mask that contains 1's in all bits up to (and including) the breakpoint and zeros afterward.
    int mask = (1 << _crossPoint) - 1;
    // Bits before the breakpoint from one parent and bits after the breakpoint from the other parent are taken.
    _child1 = (parent1.Parameter & ~mask) | (parent2.Parameter & mask);
    _child2 = (parent2.Parameter & ~mask) | (parent1.Parameter & mask);
    // Repeat the crossover if both descendants are 0
    if (_child1 == 0 && _child2 == 0)
    {
        return Crossover(parent1, parent2);
    }
    // The array is returned as the result of a crossover operation.
    var crossedGenomes = new int[2];
    crossedGenomes[0] = _child1;
    crossedGenomes[1] = _child2;
    return crossedGenomes;
}
```

# 9 Sort

The application of a genetic algorithm is intended as an expected result to obtain an optimal solution to the problem of placing the selected cargo units in the warehouse, taking into account the criteria of weight and value. For this purpose, the Sort method is implemented in the algorithm.

Cycle for (i from 0 to Selection.Count - 1) to enumerate all cargo units in the Selection list. int check = temp & 1 << i - creation of the check variable, which checks whether the bit in the i-th position in the warehouse parameter (temp) is set. if (check == (1 << i)): check the condition that the bit in the i-th position is set. If the condition is met, the cargo unit with index i is added to the list of cargo units selected for placement in the warehouse - the genome.ItemsPicked list (genom.ItemsPicked).

Thus, the Sort method looks at each bit of the genom parameter and, if the bit is set, adds the corresponding item to the genom's list of selected items. This method is used to restore the order of the selected cargo items to be placed in the warehouse based on the warehouse parameter, but taking into account the results of the sort.

```
public static void Sort(BackpackGenome genom)
    {
      //  Temporal variable temp, binary representation of the genome.
      int temp = genom.Parameter;
      //  Cycle through all items in the item list.
      for (var i = 0; i < Selection.Count; i++)
      {
        // The bit at position i in the temp variable is checked. If the bit is equal to 1, the check
will be equal to 1 << i, otherwise it will be equal to 0.
        int check = temp & 1 << i;
        // If the item has been selected (the bit at position i is 1), the item is added to the
ItemsPicked list of the genom object.
        if (check == (1 << i))
        {
          genom.ItemsPicked.Add(Selection[i]);
        }
      }
```

# 10 Breeding and Crossbreeding Through the Breaking Point, Crossover

Next, the algorithm implements its inherent evolution functions with respect to a randomly generated initial population of objects - new BackpackGenome(Rnd.Next(1, Int32.MaxValue)), but after it has been checked for adaptability and sorted.

The function returns an array that necessarily contains exactly two descendants of the genome. First, a breakpoint is defined that will be used to separate the genes of the parents. This point cannot be the first or last element to ensure that the parent and descendant genes are distinct.

Then two descendants are created. Each descendant receives a copy of the range of elements from the corresponding parent up to the breakpoint.

The loop then processes all elements from the second parent and adds them to the offspring1, in case the current gene in question is not already present in the offspring. The same is done for the offspring offspring2. Finally, the function returns an array containing the two offspring resulting from the cross.

If suddenly both descendants are 0, the crossover is executed again to avoid the situation where the descendants are completely absent. In this way, the consistency of the algorithm is ensured by the controlled evolution of the population over several generations.

```
public static int[] Crossover(BackpackGenome parent1, BackpackGenome parent2)
    {
      _crossPoint = Rnd.Next(1, Selection.Count - 1);
      // Create a mask that contains 1's in all bits up to (and including) the breakpoint and zeros
afterward.
      int mask = (1 << _crossPoint) - 1;
      // Bits before the breakpoint from one parent and bits after the breakpoint from the other
parent are taken.
      _child1 = (parent1.Parameter & ~mask) | (parent2.Parameter & mask);
      _child2 = (parent2.Parameter & ~mask) | (parent1.Parameter & mask);
      // Repeat the crossover if both descendants are 0
      if (_child1 == 0 && _child2 == 0)
      {
        return Crossover(parent1, parent2);
      }
      // The array is returned as the result of a crossover operation.
      var crossedGenomes = new int[2];
      crossedGenomes[0] = _child1;
      crossedGenomes[1] = _child2;
      return crossedGenomes;
    }
```

# 11 Mutation

Population mutation is also a variant of controlled population evolution. The method returns a new genome obtained by changing the state of a random bit. It is mandatory to check whether mutation leads to exceeding the storage capacity.

First, a random mutation point is defined, represented by the mutatePoint variable. Then a bit mask temp is created that inverts the bit at the specified point.

It is then checked to see if inverting the selected bit causes the maximum storage capacity to be exceeded. If the new genome complies with the limits, the result of the inversion is returned, otherwise the original genome is returned.

```
public static int Mutation(BackpackGenome genom)
    {
      // Index of a random item in the Selection list.
      _mutatePoint = Rnd.Next(0, Selection.Count + 1);
      // Create a mask by setting the bit according to the index.
      int temp = (1 << _mutatePoint);
      // We check whether the genome weight after mutation does not exceed the maximum
allowable weight.
      if (GetWeight(genom.Parameter ^ temp) <= MaxValue)
      {
        // Bringing back the mutated genome.
        return genom.Parameter ^ temp;
      }
      else
      {
        // Returning the original genome.
        return genom.Parameter;
      }
    }
```

## 12 Evolve

This method combines adaptation calculation, crossbreeding, mutation, and selection of the best genomes:

- items - list of genomes of the current generation.
- Solutions and NextGeneration - lists of BackpackGenome class cargo units for the current generation and the next generation.
- Cycle for (i from 0 to GENERATIONCOUNT - 1) - evolution of the genetic algorithm over several generations.
- Sort(Solutions[k]) and CalculateFitness(Solutions[k]) to sort items in a knapsack and calculate genome fitness.
- Solutions.OrderByDescending(t => t.Fitness) to sort genomes by descending fitness.
- minimalFitness - calculation of the minimum fitness value.
- While loop - iterations of the loop add genomes to the next generation through crossbreeding and mutation. Then the best genome is updated based on the adaptability of the current generation and the best genome is returned.

```
while (NextGeneration.Count < POPULATIONSIZE)
{
   // Cycle to process each genome in the current population.
   for (int m = 0; m < Solutions.Count; m++)
   {
      if (NextGeneration.Count < POPULATIONSIZE)
      {
         // Checking whether genome fitness meets the requirements to participate in
crossover.
         if (Solutions[m].Fitness >= minimalFitness && Rnd.NextDouble() <=
CROSSOVERPROBABILITY)
         {
            // If a crossover partner is not identified, the current genome becomes the
partner.
            if (_crossoverPartner == null)
            {
               _crossoverPartner = Solutions[m];
            }
            // If a partner has already been identified, crossover between the current
genome and the partner occurs.
            else
            {
               _crossedGenomes = Crossover(_crossoverPartner, Solutions[m]);
               // Create new genomes based on values from the array
_crossedGenomes.
               // Going through each new genome.
               foreach (var genome in _crossedGenomes.Select(child => new
BackpackGenome(child)))
               {
                  // If the weight of a new genome does not exceed the maximum
allowable weight, that genome is added to the next generation.
                  if (GetWeight(genome.Parameter) <= MaxValue)
                  {
                     NextGeneration.Add(genome);
                  }
               }
            }
         }
         // Probability of genome mutation.
         if (Rnd.NextDouble() <= MUTATIONPROBABILITY)
         {
            NextGeneration.Add(new BackpackGenome(Mutation(Solutions[m])));
         }
```

Before the function is completed, the output of the received data is performed. The user gets all the information he needs: value and weight of each cargo unit in the warehouse, maximum weight, current weight, current value, the weight of all cargo units, and value of all cargo units.

Output results to the console.

```
        generationCallback?.Invoke("Selected items:");
        generationCallback?.Invoke("");
        foreach (BackpackItem t in Result.ItemsPicked)
        {
           generationCallback?.Invoke(t.Name + "  " + "\t" + " (Value:" + t.Worth + ") " +
                 "\t" + "(Bec:" + t.Weight + ")");
        }
        generationCallback?.Invoke("");
        generationCallback?.Invoke("Maximum weight: " + MaxValue);
        generationCallback?.Invoke("Current weight: " + Result.ItemsPicked.Sum(t =>
t.Weight));
        generationCallback?.Invoke("Current value: " + Result.ItemsPicked.Sum(t => t.Worth));
        generationCallback?.Invoke("");
        generationCallback?.Invoke("Weight of all items: " + Selection.Sum(t => t.Weight));
        generationCallback?.Invoke("The value of all items: " + Selection.Sum(t => t.Worth));
    }
```

As a result, the algorithm selects the most adaptive placement solutions for the selected cargo units at each step in the evolution of the generated random solutions to result in an optimal placement solution.

## 13 Application Architecture

The application is divided into several parts with different areas of responsibility.

Interaction with the menu and initialization of instances of classes that solve the tasks is performed in Program.cs.

The BackpackSolver class is responsible for solving the problem of filling the warehouse with an optimal selection of cargo units. For more convenient data visualization and structuring of calculations, BackpackGenome, which stores information about the warehouse, and BackpackItem, which represents a cargo unit in the warehouse, are also used.

The application uses:

- interface. The BackpackSolver class implements the IGeneticAlgorithm interface.
- Delegates. The GenerationCallback delegate is used to output the program operation data, which is passed when class instances are initialized.
- exceptions. Exceptions are used to handle incorrect user input to better explain the error.
- Enumerations. MenuOption.cs is an enumeration of menu states that is used to trigger the necessary actions when an item is selected.
- LINQ. LINQ is used to simplify the handling of collections for sorting, mutation, and other operations in BackpackSolver.

## 14 Machine Experiment

To develop and test the program that implements the optimal solution for placing the selected cargo units, an experiment was conducted. An experiment plan with the definition of parameters, testing scenario, and expected results was made in advance. When the program is run, the problem is solved using the genetic algorithm, or the program is exited. The purpose of the experiment included a) checking the correctness of data input by the user; b) checking the correctness of error case processing; c) evaluating the functionality of the program at different data inputs.

The input data included: a) warehouse capacity: positive integer; b) number of cargo units: positive integer; c) for each cargo unit weight: positive integer (can be integer or fractional); d) for each cargo unit value (equivalent to placement cost): positive integer (can be integer or fractional).

Error conditions were defined as follows: a) input of negative numbers or zero for capacity and number of cargo units; b) incorrect input format (e.g. letters instead of numbers); c) weight and cost of cargo units should be positive. The testing scenario included: a) testing of correct input; b) testing of incorrect input for capacity; c) testing of incorrect input for number of cargo units; d) testing of incorrect input for weight and cost; e) testing on boundary values.

The following methods of results evaluation were chosen: a) checking the presence of error messages in case of incorrect input; b) confirmation of successful program completion in case of correct data input; c) analysis of program response time to user input.

The obtained results of the test scenario implementation corresponded to the expected results:

a) the program successfully accepts input without errors; b) in case of incorrect input - an error message and a repeated request. No changes in the program were required after the test scenario was executed.

The program implements the developed algorithm as follows: it requests from the user the warehouse capacity and the number of cargo units, then after receiving this information the program requests information on each cargo unit in turn: weight and cost. In case of non-compliance with the task conditions (entering a smaller number or an incorrect format), the corresponding error is displayed and the quantity is requested again.

The test parameters took the following values:

1. Warehouse capacity: minimum value: 0 (a warehouse cannot have a negative capacity); maximum value: set depending on the specifics of the task (e.g., 1000 units).
2. Number of cargo units: minimum value: 1 (there must be at least one cargo unit); maximum value: 100 cargo units.
3. Weight of cargo units: minimum value: 0.1 (cargo cannot have zero weight); maximum value: 500 kg.
4. Value of cargo units: minimum value: 0 (the value cannot be negative); maximum value: 100000 conventional monetary units.

The parameters of the experiment included the following test sample (variances):

1. Population size: population size of 50 to 200 individuals to ensure diversity.
2. Number of generations: 100 to 500 to ensure sufficient time for evolution of solutions.
3. Mutation percentage: 5-10% to avoid premature convergence and maintain genetic diversity.
4. Inbreeding percentage: at 70-90% to ensure an efficient combination of the best solutions.
5. Elitism: 2 to 5% of the best-adapted individuals to be passed on to the next generation unchanged.
6. Algorithm stopping criteria: a) stopping when the maximum number of generations is reached; b) stopping when a given level of adaptability is reached (if the best individual achieves fitness above a certain threshold); c) stopping when there is no significant change for several generations (20 generations).

When correct values were entered, the program worked: the most suitable solutions for placing the selected cargo units were selected. The selected cargo units (their value and weight), maximum warehouse capacity, weight, and value of all cargo units were displayed.

The warehouse operator uses the obtained data to decide on the order of filling the warehouse space.

## 15 Conclusion

The genetic algorithm has shown its ability to process data, as well as demonstrated a high rate of convergence to optimal solutions in comparison with classical oversampling methods. When selecting the genetic algorithm as a mathematical apparatus for solving the problem of placement in the warehouse of a multimodal transport and logistics center of an optimal selection of cargo units, their flexibility and applicability to a variety of problems were revealed. As a result, it was possible to obtain a software tool for effective management of the placement of cargo

units taking into account their weight and value and under conditions of limited warehouse capacity. This creates opportunities for practical application of the proposed results and makes them promising for solving a wide range of optimization problems in the warehouse management system.

The possibilities for further development in this area are the deepening of research in optimizing the parameters of genetic algorithms to improve their efficiency and accuracy on different types of warehouse management problems. It is also worth considering combining genetic algorithms with other optimization methods to create more efficient hybrid approaches. In addition, research in parallel and distributed computing is expected to significantly improve the speed of genetic algorithms and their applicability to even more complex multimodal transportation and logistics center optimization problems.

## Declaration of Generative AI and AI-assisted Technologies in the Writing Process

During the preparation of this work the authors used Perplexity.AI in order to correct errors and optimize machine code. After using this tool/service, the authors reviewed and edited the content as needed and takes full responsibility for the content of the publication.

*References:*

[1] Sun J., Wang R., Multi-objective optimization of a sustainable two echelon vehicle routing problem with simultaneous pickup and delivery in construction projects, *Journal of Engineering Research*, In Press, 2023, https://doi.org/10.1016/j.jer.2023.10.033.

[2] Liu W., Zhou Y., Liu W., Qiu J., Xie N., Chang X., Chen J., A hybrid ACS-VTM algorithm for the vehicle routing problem with simultaneous delivery & pickup and real-time traffic condition, *Computers & Industrial Engineering*, Vol. 162, 107747, 2021, https://doi.org/10.1016/j.cie.2021.107747.

[3] Praxedes R., Bulhões T., Subramanian A., Uchoa E., A unified exact approach for a broad class of vehicle routing problems with simultaneous pickup and delivery, *Computers & Operations Research*, Vol. 162, 106467, 2024, https://doi.org/10.1016/j.cor.2023.106467.

[4] Liu S., Tang K., Yao X., Memetic search for vehicle routing with simultaneous pickup-delivery and time windows, *Swarm and Evolutionary Computation*, Vol. 66, 100927, 2021, https://doi.org/10.1016/j.swevo.2021.100927.

[5] Grabusts P., Musatovs J., Golenkov V., The application of simulated annealing method for optimal route detection between objects, *Procedia Computer Science*, Vol. 149, pp.95-101, 2019, https://doi.org/10.1016/j.procs.2019.01.112.

[6] Ergüven E., Polat F., Relative distances approach for multi-traveling salesmen problem, *Knowledge-Based Systems*, Vol. 300, 112160, 2024, https://doi.org/10.1016/j.knosys.2024.1121.

[7] Roy A., Maity S., Moon I., Multi-vehicle clustered traveling purchaser problem using avariable-length genetic algorithm, *Engineering Applications of Artificial Intelligence*, Vol. 123, Part B, 106351, 2023, https://doi.org/10.1016/j.engappai.2023.106351.

[8] Yu Y., Machemehl R.B., Xie Ch., Demand-responsive transit circulator service network design, *Transportation Research Part E: Logistics and Transportation Review*, Vol. 76, pp.160-175, 2015, https://doi.org/10.1016/j.tre.2015.02.009.

[9] Kerbache L., Smith J.MacG., Multi-objective routing within large scale facilities using open finite queueing networks, *European Journal of Operational Research*, Vol. 121, Issue1, pp.105-123, 2000, https://doi.org/10.1016/S0377-2217(99)00018-1.

[10] Prokofieva T.A., Multimodal transport and logistics centers as strategic points of growth of the Russian economy, Part 1, *In Center of Economics*, no. 2, pp. 10-19, 2021.

[11] Park Y.M., Park J.B., Won J.R., A hybrid genetic algorithm/dynamicprogramming approach to optimal long-term generation expansion planning, International Journal of Electrical Power & Energy Systems, Volume 20, Issue 4, 1998, pp.295-303, https://doi.org/10.1016/S0142-0615(97)00070-7.

[12] Wang Q.J., Using genetic algorithms to optimise model parameters, Environmental Modelling & Software, Vol. 12, Issue 1, 1997, pp.27-34, https://doi.org/10.1016/S1364-

8152(96)00030-8.
[13] Chen J., Zhang Ch., Efficient Clustering Method Based on Rough Set and Genetic Algorithm, Procedia Engineering, Vol.15, 2011, pp.1498-1503, https://doi.org/10.1016/j.proeng.2011.08.278.
[14] Kochenderfer M. J., Wheeler T. A., Algorithms for Optimization, MIT Press, 2019.
[15] Horn J., Goldberg D. E., Genetic Algorithm Difficulty and the Modality of Fitness Landscapes, *Foundations of Genetic Algorithms*, Elsevier, Vol. 3, 1995, pp. 243-269, https://doi.org/10.1016/B978-1-55860-356-1.50016-9.
[16] Kohenderfer M., Wheeler T., Ray K. *Algorithms for decision making* / transl. from Engl. V. S. Yatsenkov, Moscow, DMK Press, p.684, 2023.
[17] Karp R. M., Reducibility among Combinatorial Problems, In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds) Complexity of Computer Computations, *The IBM Research Symposia Series*, Springer, Boston, MA, 1972, https://doi.org/10.1007/978-1-4684-2001-2_9.
[18] D'Angelo G., Palmieri F., GGA: A modified genetic algorithm with gradient-based local search for solving constrained optimization problems, *Information Sciences*, Vol. 547, 2021, pp.136-162, https://doi.org/10.1016/j.ins.2020.08.040.
[19] Dao S. D., Abhary K., Marian R., An innovative framework for designing genetic algorithm structures, Expert Systems with Applications, Vol. 90, 2017, pp.196- 208, https://doi.org/10.1016/j.eswa.2017.08.018.
[20] Mahmoodabadi M.J., Nemati A.R., A novel adaptive genetic algorithm for global optimization of mathematical test functions and real-world problems, *Engineering Science and Technology, an International Journal*, Vol. 19, Issue 4, 2016, pp. 2002-2021, https://doi.org/10.1016/j.jestch.2016.10.012.
[21] Martello S., Toth P., *Knapsack problems: algorithms and computer implementations.* John Wiley & Sons, Inc., USA, 296 p., 1990.
[22] Wilbaut C., Hanafi S., Coelho I.M., Lucena A., *The Knapsack Problem and Its Variants: Formulations and Solution Methods.* In: Salhi, S., Boylan, J. (eds) The Palgrave Handbook of Operations Research, Palgrave Macmillan, Cham, pp 105-151, 2022, https://doi.org/10.1007/978-3-030-96935-6_4.
[23] Reeves C.R., Genetic Algorithms. In: LIU, L., ÖZSU, M.T. (eds) *Encyclopedia of DatabaseSystems*, Springer, Boston, MA, 2009, https://doi.org/10.1007/978-0-387-39940- 9_562.
[24] Holland J.H., GeneticAlgorithms and Adaptation. In: Selfridge, O.G., Rissland, E.L., Arbib, M.A. (eds) Adaptive Control of Ill-Defined Systems, NATO Conference Series, vol 16, Springer, Boston, MA, 1984, https://doi.org/10.1007/978-1-4684-8941- 5_21.
[25] Hnaien F., Delorme X., Dolgui A., Genetic algorithm for supply planning in two-level assembly systems with random lead times, *Engineering Applications of Artificial Intelligence*, Vol. 22, Issue 6, Pages 906-915, 2009, https://doi.org/10.1016/j.engappai.2008.10.012.

**Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)**

The author equally contributed to the present research, at all stages from the formulation of the problem to the final findings and solution.

**Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself**

The research was funded by the grant Russian Science Foundation No.24-21-20089.

**Conflict of Interest**

The authors have no conflicts of interest to declare.