

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

# Метод упаковки контейнеров в трюм корабля на основе генетического алгоритма

Студент	ИУ7-84Б		Д. Тумайкина
	(группа)	(подпись)	(инициалы, фамилия)
			Т. Н. Романова
		(подпись)	(инициалы, фамилия)
		(подпись)	(инициалы, фамилия)

2025 год

## РЕФЕРАТ

Расчетно-пояснительная записка 26 рис., 66 с., 3 табл., 18 ист., 1 прил.

Ключевые слова: ТРЮМ, КОНТЕЙНЕР, ЗАДАЧА ТРЕХМЕРНОЙ УПАКОВКИ, ОПТИМИЗАЦИЯ, ЭВРИСТИЧЕСКИЕ АЛГОРИТМЫ, ГЕНЕТИЧЕСКИЙ АЛГОРИТМ.

Целью данной выпускной квалификационной работы является разработка метода упаковки контейнеров в трюм корабля на основе генетического алгоритма и разработка программного обеспечения, реализующего данный метод.

В аналитическом разделе проведен анализ предметной области, формализованы критерии и ограничения оптимизации упаковки контейнеров в трюм корабля. Формализована постановка задачи в виде математической модели. Проведен обзор существующих эвристических методов для решения оптимизационных задач, приведен результат сравнительного анализа.

В конструкторском разделе модифицирован генетический алгоритм для решения поставленной задачи, описаны входные и выходные данные разрабатываемого метода, представлены ключевые этапы метода в виде схем алгоритмов. Описаны компоненты программы и их взаимодействие.

В технологическом разделе обоснован выбор программных средств реализации предложенного метода, описан формат входных и выходных данных и структура разрабатываемого ПО. Реализован метод упаковки контейнеров в трюм корабля, описано взаимодействие пользователя с программным обеспечением.

В исследовательском разделе проверена адекватность построенной модели, исследована эффективность реализованного метода при различных значениях параметров.

# СОДЕРЖАНИЕ

<b>РЕФЕРАТ</b>	<b>5</b>
<b>ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ</b>	<b>8</b>
<b>ВВЕДЕНИЕ</b>	<b>9</b>
<b>1 Аналитический раздел</b>	<b>10</b>
1.1 Анализ предметной области . . . . .	10
1.1.1 Описание основных понятий . . . . .	10
1.2 Актуальность задачи . . . . .	11
1.3 Постановка задачи . . . . .	12
1.4 Математическая модель . . . . .	12
1.5 Обзор методов оптимизации . . . . .	14
1.5.1 Жадные алгоритмы . . . . .	14
1.5.2 Алгоритм имитации отжига . . . . .	15
1.5.3 Генетический алгоритм . . . . .	16
1.5.4 Алгоритмы муравьиных колоний . . . . .	17
1.6 Сравнение методов . . . . .	18
1.7 Вывод . . . . .	19
<b>2 Конструкторский раздел</b>	<b>21</b>
2.1 Разработка алгоритма . . . . .	21
2.1.1 IDEF0 диаграмма первого уровня . . . . .	21
2.1.2 Схема генетического алгоритма . . . . .	22
2.1.3 Кодирование особей . . . . .	24
2.1.4 Генерация начальной популяции . . . . .	24
2.1.5 Вычисление фитнес-функции . . . . .	26
2.1.6 Отбор особей . . . . .	31
2.1.7 Кроссовер . . . . .	33
2.1.8 Мутация . . . . .	35
2.2 Разработка архитектуры программного обеспечения . . . . .	37
2.3 Вывод . . . . .	38

<b>3</b>	<b>Технологический раздел</b>	<b>39</b>
3.1	Выбор средств программной реализации . . . . .	39
3.2	Формат входных и выходных данных . . . . .	39
3.3	Реализация алгоритма упаковки контейнеров в трюм в заданном порядке . . . . .	42
3.4	Реализация генетического алгоритма . . . . .	45
3.5	Демонстрация работы программы . . . . .	50
3.6	Вывод . . . . .	52
<b>4</b>	<b>Исследовательский раздел</b>	<b>54</b>
4.1	Проведение исследований . . . . .	54
4.2	Проверка адекватности построенной модели . . . . .	54
4.3	Исследование зависимости сходимости от процента элитизма и ко- личества особей в турнире . . . . .	58
4.4	Вывод . . . . .	61
	<b>ЗАКЛЮЧЕНИЕ</b>	<b>62</b>
	<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>63</b>
	<b>ПРИЛОЖЕНИЕ А</b>	<b>66</b>

## ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей выпускной квалификационной работе используются следующие обозначения и сокращения:

- **EMS** — максимальное свободное пространство;
- **ПО** — программное обеспечение;
- **UI** — интерфейс пользователя;
- **ООП** — объектно-ориентированное программирование.

## ВВЕДЕНИЕ

В условиях глобализации и быстрого роста международной торговли объем морских грузоперевозок постоянно увеличивается. Это делает задачу оптимизации логистических процессов и транспортировки грузов особенно актуальной для судоходных компаний, логистических операторов и производителей. Эффективное управление грузопотоками, рациональное использование транспортных ресурсов и оптимальная загрузка трюмов судов позволяют снизить затраты на перевозки, повысить производительность логистических операций и минимизировать воздействие на окружающую среду. При решении данной задачи важно учитывать размеры контейнеров и трюма судна.

Целью данной работы является разработка метода упаковки контейнеров в трюм корабля на основе генетического алгоритма и разработка программного обеспечения, реализующего данный метод.

Достижение поставленной цели требует решения следующих задач:

- выбрать критерии и ограничения оптимизации упаковки контейнеров в трюм корабля, формализовать постановку задачи в виде математической модели;
- провести обзор существующих эвристических методов для решения оптимизационных задач, привести результаты сравнительного анализа;
- модифицировать генетический алгоритм для решения поставленной задачи, представить ключевые этапы метода в виде схем алгоритмов;
- создать программную реализацию разработанного метода;
- проверить адекватность построенной модели, исследовать эффективность реализованного метода при различных значениях параметров.

## **1 Аналитический раздел**

### **1.1 Анализ предметной области**

#### **1.1.1 Описание основных понятий**

Трюмы кораблей и контейнеры являются ключевыми элементами в современной морской логистике.

Трюм — это внутреннее пространство корабля, предназначенное для размещения грузов. Его форма, размеры и конструкция зависят от типа судна и вида перевозимого груза. Прямоугольные трюмы чаще всего используются на контейнеровозах, где груз перевозится в стандартных морских контейнерах.

Контейнеры — это стандартизированные упаковки, используемые для транспортировки грузов. Наиболее распространены прямоугольные контейнеры, которые подразделяются на несколько типов в зависимости от их назначения [1]:

- секционные контейнеры: используются для перевозки генеральных грузов, таких как товары народного потребления, оборудование и промышленные материалы (стандартные размеры включают 20-футовые - TEU, и 40-футовые контейнеры - FEU);
- рефрижераторные контейнеры: предназначены для перевозки скоропортящихся грузов, таких как продукты питания, фармацевтические препараты и химикаты, требующие контроля температуры;
- танк-контейнеры: используются для перевозки жидкостей, включая топливо, воду и химические вещества;
- открытые контейнеры: применяются для крупногабаритных грузов, которые не могут быть упакованы в стандартные контейнеры.

Контейнеровозы делятся на несколько категорий в зависимости от того, сколько стандартных контейнеров могут вместить [2]:

1. Ультрабольшие контейнеровозы - суда с вместимостью более 14000 TEU. На сегодняшний день построено лишь несколько таких судов, так как слишком велики для прохождения через любые каналы.

2. Пост-Панамакс - суда, которые слишком крупны для прохождения через текущий Панамский канал и предназначены для трансконтинентальных рейсов. Их вместимость обычно составляет 5500-8000 TEU, хотя были построены и более крупные суда с вместимостью свыше 10000 TEU.
3. Новый Панамакс - суда, которые смогут проходить через расширенный Панамский канал. Их вместимость может достигать около 12000 TEU.
4. Панамакс - суда, которые могут проходить через текущий Панамский канал. Их вместимость составляет от 1000 до 5000 TEU, а количество трюмов - от 5 до 7.
5. Фидерные суда (Feeder ships) - небольшие суда, которые не совершают океанские рейсы, а обычно занимаются перевозкой контейнеров на короткие расстояния. Могут перевозить несколько сотен TEU. Обычно имеют до 3 трюмов.

В данной работе рассматривается оптимизация загрузки контейнеров в фидерное судно.

## **1.2 Актуальность задачи**

Задача оптимизации загрузки контейнеров в трюм корабля является важным аспектом современной морской логистики. Морские перевозки остаются основным способом транспортировки грузов на большие расстояния. По данным экспертов UNCTAD - Конференции Организации объединенных наций по торговле и развитию, до 80% объема мировой торговли перевозится морским путем и обрабатывается в портах по всему миру [3]. В связи с этим возникает потребность в повышении эффективности перевозок.

Не менее важным является факт, что контейнеровозы значительно увеличились в размерах за последние 20 лет. Если в 2002 году крупный контейнеровоз мог перевозить примерно 6500 TEU (двадцатифутовых контейнеров), то сегодня самые большие контейнеровозы способны перевозить почти 24000 единиц [4].

Таким образом, задача оптимизации упаковки контейнеров в трюм корабля особенно актуальна в наше время, так как позволяет повысить экономическую эффективность и адаптироваться к требованиям глобальной кон-



курении [5]. Благодаря внедрению современных технологий и подходов, данная задача становится все более решаемой, что делает ее важным направлением исследований и разработок в сфере транспортировки.

### 1.3 Постановка задачи

Оптимизационная задача упаковки контейнеров в трюм корабля ставится следующим образом. Имеется трюм и набор контейнеров, которые необходимо в него упаковать (до 200 контейнеров). Трюм характеризуется его габаритами и грузоподъемностью, контейнер - размерами и весом. Необходимо упаковать контейнеры в трюм так, чтобы минимизировать свободное пространство в нем. Трюм и контейнеры имеют форму параллелепипеда.

Поставленная задача имеет ряд ограничений:

- упакованные контейнеры не могут выходить за границы трюма;
- упакованные контейнеры не могут пересекаться, то есть два контейнера не могут занимать одну и ту же область пространства;
- суммарный объем упакованных контейнеров не должен превышать объем трюма;
- суммарный вес упакованных контейнеров не должен превышать грузоподъемность трюма;
- все контейнеры должны упираться основанием на поверхность трюма или другие контейнеры.

### 1.4 Математическая модель

Решение задачи упаковки контейнеров в трюм корабля можно свести к математической задаче трехмерной упаковки с ограниченной высотой области упаковки [6][7].

Опишем трюм судна как область трехмерного пространства шириной  $W$ , длиной  $L$  и высотой  $H$ :

$$M = L \times W \times H. \quad (1.1)$$

У трюма также есть грузоподъемность  $G$ .

Все контейнеры представляют собой множество из  $N$  блоков. Каждый контейнер описывается кортежем из 4 элементов:

$$\{l_i, w_i, h_i, g_i\}, \quad (1.2)$$

где  $l_i, w_i, h_i, g_i$  – длина, ширина, высота и вес контейнера соответственно ( $i = \overline{1, N}$ ).

Упакованные в трюм контейнеры представляют собой множество из  $P$  блоков. Расположение каждого упакованного контейнера в трюме описывается кортежем вида:

$$\{x_{0i}, y_{0i}, z_{0i}, x_{1i}, y_{1i}, z_{1i}\}, \quad (1.3)$$

где  $x_{0i}, y_{0i}, z_{0i}$  – координаты расположения самого близкого к началу координат угла,  $x_{1i}, y_{1i}, z_{1i}$  – координаты расположения самого дальнего от начала координат угла ( $i = \overline{1, P}$ ).

Задача заключается в том, чтобы разместить множество контейнеров в заданный объем трюма. Выходными данными будут являться координаты упакованных контейнеров в трюме и значение целевой функции (ЦФ).

Данная задача подразумевает следующие ограничения:

1. Ни один упакованный контейнер не может выходить за границы трюма:

$$\left\{ \begin{array}{l} x_{0i} \geq 0, \\ y_{0i} \geq 0, \\ z_{0i} \geq 0, \\ x_{1i} \leq L, \\ y_{1i} \leq W, \\ z_{1i} \leq H, i = \overline{1, P}. \end{array} \right. \quad (1.4)$$

2. Суммарный объем упакованных контейнеров не должен превышать объем трюма:

$$\sum_{i=1}^P (l_i \cdot w_i \cdot h_i) \leq L \cdot W \cdot H. \quad (1.5)$$

3. Суммарный вес упакованных контейнеров не должен превышать грузоподъемность трюма:

$$\sum_{i=1}^P g_i \leq G. \quad (1.6)$$

4. Упакованные контейнеры не должны пересекаться:

$$\begin{aligned} &((x_{0i} \geq x_{1j}) \vee (x_{1i} \leq x_{0j}) \vee (y_{0i} \geq y_{1j}) \vee \\ &(y_{1i} \leq y_{0j}) \vee (z_{0i} \geq z_{1j}) \vee (z_{1i} \leq z_{0j}) = 1, \\ &\forall i \leq P; \forall j \leq P; i \neq j. \end{aligned} \quad (1.7)$$

Критерием оптимизации является суммарный объем, занимаемый упакованными контейнерами. Целевая функция имеет вид:

$$F = V_{hold} - \sum_{i=1}^P V_i \rightarrow \min, \quad (1.8)$$

где  $V_{hold}$  – объем трюма,  $V_i$  – объем упакованного контейнера.

Это означает, что необходимо стремиться к уменьшению пустого пространства в трюме.

## 1.5 Обзор методов оптимизации

Задача трехмерной упаковки относится к классу NP-сложных задач, то есть ее не решить за полиномиальное время при больших объемах данных. Поэтому для решения данной проблемы используются не точные, а приближенные методы.

Эвристические методы — это подходы к решению задач оптимизации, которые не гарантируют нахождение самого оптимального решения, но способны быстро находить достаточно хорошие результаты. Они широко используются в задачах упаковки контейнеров из-за своей простоты и эффективности при работе с большими объемами данных.

### 1.5.1 Жадные алгоритмы

Жадный алгоритм — это метод решения оптимизационных задач, который в каждом шаге выбирает локально оптимальное решение, с намерением достичь глобального оптимума. Принцип работы жадного алгоритма основывается на следующем: на каждом шаге выбирается наилучший возможный

вариант среди доступных, при этом не учитываются будущие последствия выбора[8].

Работа алгоритма начинается с выбора объекта, который следует разместить первым, основываясь на его размере или других характеристиках, например, по объему или наибольшей стороне. Далее объекты размещаются по принципу "первый пришел — первый упакован" что позволяет избежать сложных вычислений, связанных с динамическим распределением. Алгоритм пытается оптимизировать пространство, начиная с угла или центра контейнера и постепенно добавляя объекты, размещая их в пустые зоны. При размещении каждого нового объекта проверяется, можно ли его поместить в доступную область без пересечений с уже размещенными элементами. При этом выбирается наилучшее место для упаковки, например, с учетом минимизации пустых промежутков.

Главным достоинством жадных алгоритмов является их простота и высокая скорость работы. Однако он может не всегда приводить к оптимальному решению, так как не учитывает глобальных характеристик упаковки. Алгоритм часто выбирает локально оптимальные решения, что может не быть оптимальным в глобальном контексте.

### **1.5.2 Алгоритм имитации отжига**

Алгоритм имитации отжига (Simulated Annealing, SA) — это стохастический метод оптимизации, вдохновленный процессом охлаждения материала, при котором атомы перестраиваются в более низкоэнергетическое состояние. Алгоритм начинается со случайного решения задачи и постепенно "охлаждает" систему, уменьшая вероятность принятия менее оптимальных решений. На каждом шаге алгоритм генерирует новое решение, которое близко к текущему, и оценивает его с помощью функции стоимости. Если новое решение лучше, оно принимается; если хуже — оно может быть принято с определенной вероятностью, которая зависит от температуры[9].

Температура — это параметр, который управляет вероятностью принятия худших решений. Сначала температура высока, что позволяет алгоритму исследовать более широкий набор решений и избегать застревания в локальных оптимумах. По мере уменьшения температуры вероятность принятия плохих решений снижается, и алгоритм начинает сосредотачиваться на поиске глобального оптимума.

Процесс "охлаждения" в алгоритме происходит по заранее заданному графику, обычно экспоненциально, что означает постепенное уменьшение температуры с каждым шагом. Алгоритм продолжается до тех пор, пока температура не станет достаточно низкой или пока не будет достигнуто заданное количество итераций.

Основное преимущество алгоритма имитации отжига заключается в его способности избегать локальных минимумов, что позволяет ему находить более эффективные решения по сравнению с простыми жадными методами. Однако его недостатком является то, что алгоритм требует настройки параметров (например, начальной температуры, скорости охлаждения), что может влиять на качество и скорость нахождения решения.

### **1.5.3 Генетический алгоритм**

Генетический алгоритм — это метод оптимизации, основанный на принципах естественного отбора и генетики. Его цель — найти наилучшее решение задачи путём имитации процессов, происходящих в живых организмах. Основная идея заключается в эволюционном улучшении популяции решений с течением времени [10].

Алгоритм начинается с создания случайной популяции решений, называемых индивидами. Каждый индивид представлен хромосомой, которая может быть строкой чисел или других значений, отражающих параметры решения задачи. Затем каждому индивиду присваивается значение функции приспособленности, которое оценивает качество решения. Эту функцию можно настроить в зависимости от целей задачи. Чем выше приспособленность, тем лучше индивид подходит для решения проблемы.

Следующий шаг — это отбор родителей для создания нового поколения. Отбор происходит на основе их приспособленности, при этом лучшие особи имеют больше шансов быть выбранными для скрещивания.

После отбора происходит скрещивание (или кроссовер), где части хромосом родителей комбинируются для создания потомков. Этот процесс имитирует половое размножение в природе. Скрещивание позволяет передавать лучшие черты родителей и комбинировать их для получения новых решений. Далее идет этап мутации, где случайным образом изменяются некоторые части хромосом потомков. Мутация добавляет случайность в процесс и помогает избегать застревания в локальных минимумах.

Новая популяция потомков оценивается с точки зрения их приспособленности, и лучшие из них выбираются для дальнейшего поколения. Это повторяется на протяжении заданного числа поколений, при этом каждый раз популяция должна становиться лучше. Иногда используется стратегия эволюционного элитизма, когда лучшие особи из текущего поколения сохраняются без изменений и передаются в следующее поколение.

Процесс повторяется до тех пор, пока не будет достигнуто условие завершения, которое может быть основано на максимальном числе поколений, достижении приемлемого уровня приспособленности или других критериях.

Алгоритм имеет несколько важных параметров: размер популяции, вероятность скрещивания и мутации, а также количество поколений. Эти параметры могут быть настроены для оптимизации работы алгоритма в зависимости от задачи.

Сильной стороной ГА является способность работать с большими и сложными пространствами решений, где традиционные методы могут не подойти. Алгоритм также способен учитывать множество ограничений и критических факторов, что делает его особенно полезным в многокритериальных задачах. Однако, чтобы добиться хороших результатов, необходимо тщательно настроить параметры алгоритма и правильно выбрать функцию приспособленности.

#### **1.5.4 Алгоритмы муравьиных колоний**

Муравьиный алгоритм (Ant Colony Optimization, ACO) — это эвристический алгоритм оптимизации, вдохновленный поведением реальных муравьев при поиске пищи и построении оптимальных путей. Алгоритм имитирует коллективную работу муравьев, которые, оставляя феромоны на своем пути, помогают другим муравьям находить лучшие маршруты. Каждый муравей представляет собой агента, который перемещается по графу, представляющему пространство возможных решений[11].

Начальная популяция муравьев размещается случайным образом, и они начинают искать пути по графу. Когда муравей проходит по пути, он оставляет на нем феромоны, причем количество феромонов пропорционально качеству найденного решения. С увеличением числа муравьев, которые идут по тому же пути, концентрация феромонов на этом пути увеличивается, и другие муравьи начинают более часто выбирать этот путь, так как он кажется

им более привлекательным.

Каждый муравей выбирает путь к следующему элементу на основе феромонов и расстояния. Феромоны обновляются по следующей формуле:

$$\tau_{ij}(t+1) = (1 - \rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t), \quad (1.9)$$

где  $\tau_{ij}(t)$  — количество феромонов на пути  $i \rightarrow j$  на момент времени  $t$ ;  $\rho$  — коэффициент испарения феромонов;  $\Delta\tau_{ij}(t)$  — изменение феромонов, зависящее от качества найденного пути:

$$\Delta\tau_{ij}(t) = \sum_{k=1}^N \frac{Q}{L_k}, \quad (1.10)$$

где  $Q$  — константа,  $L_k$  — длина пути, пройденного муравьем  $k$ .

Множество муравьев, находящихся в одной популяции, проходит через все возможные маршруты, и наибольшее количество феромонов остается на наиболее коротких путях. По мере выполнения алгоритма феромоны на менее оптимальных путях испаряются, что снижает вероятность их выбора. Затем процесс повторяется в новых поколениях. На каждом шаге алгоритм использует вероятность  $p_{ij}(t)$  для выбора пути, который муравей будет следовать. Чем больше феромонов на пути, тем выше вероятность его выбора. Если решение достигло заранее установленного критерия (например, минимальная длина пути или минимальная ошибка), то алгоритм завершает свою работу.

Результат работы алгоритма зависит от множества параметров, таких как коэффициенты  $\alpha$ ,  $\beta$  и  $\rho$ . Гибкость алгоритма позволяет применить его в самых разных задачах, от маршрутизации до упаковки и проектирования.

## 1.6 Сравнение методов

Для сравнения вышеописанных методов были выделены следующие критерии:

1. Решение за полиномиальное время - в задаче упаковки контейнеров в трюм корабля используется большое количество контейнеров, поэтому необходимо, чтобы метод находил решение за полиномиальное, а не экспоненциальное, время;
2. Устойчивость к нахождению локального оптимума - важно, чтобы ал-

горитм не "застревал" на нахождении решения, которое лучше, чем все решения в окрестности, но не является достаточно хорошим на глобальном уровне;

3. Возможность параллельной обработки - задача упаковки контейнеров в трюм корабля требует вычисления больших объемов данных, и для более быстрого нахождения решения важна возможность распараллеливания алгоритма;

В таблице 1 представлено сравнение рассмотренных методов для решения задачи упаковки контейнеров в трюм корабля:

Таблица 1 – Сравнение методов

Метод	Решение за полиномиальное время	Устойчивость к нахождению локального оптимума	Возможность параллельной обработки
Жадный	+	-	+
Имитации отжига	+	+	-
Генетический	+	+	+
Муравьиной колонии	+	+	+

Среди рассмотренных методов лишь генетический алгоритм и алгоритм муравьиной колонии удовлетворяют всем заданным критериям. Для обоих методов необходима тщательная настройка ключевых параметров, которые могут сильно влиять на эффективность и результаты работы алгоритма. Однако, параметры генетического алгоритма интуитивно понятны и более просты в настройке в сравнении с муравьиным. Поэтому для решения поставленной задачи упаковки контейнеров в трюм корабля решено отдать предпочтение генетическому алгоритму.

## 1.7 Вывод

В данном разделе был проведен анализ предметной области, сформулирована постановка задачи и ее математическая модель, выбраны и формализованы критерии и ограничения оптимизации, разработана целевая функция. Был проведен обзор эвристических алгоритмов оптимизации, сформулиро-



ваны критерии сравнения методов. В результате проделанной работы для решения поставленной задачи был выбран генетический алгоритм.

## 2 Конструкторский раздел

### 2.1 Разработка алгоритма

#### 2.1.1 IDEF0 диаграмма первого уровня

Метод решения поставленной задачи представляет собой комбинацию двух взаимосвязанных алгоритмов. Генетический алгоритм работает на уровне генерации и эволюции популяции и отвечает за поиск оптимального решения. Для вычисления фитнес-функции и проверки выполнения ограничений необходим внутренний алгоритм упаковки, который определяет размещение контейнеров для конкретной особи в популяции. Таким образом, генетический алгоритм отвечает за стохастический поиск в пространстве решений, а внутренний метод упаковки обеспечивает корректность и оценивает качество каждой особи.

Рисунок 1 демонстрирует схему работы генетического алгоритма в виде функциональной модели IDEF0. К параметрам генетического алгоритма относятся:

- количество особей в популяции;
- количество поколений;
- процент мутации;
- процент элитных особей в популяции;
- количество особей в турнире.

На вход алгоритма также подаются параметры трюма корабля - габариты и грузоподъемность, и параметры контейнеров - массив с id, габаритами и весом каждого контейнера.

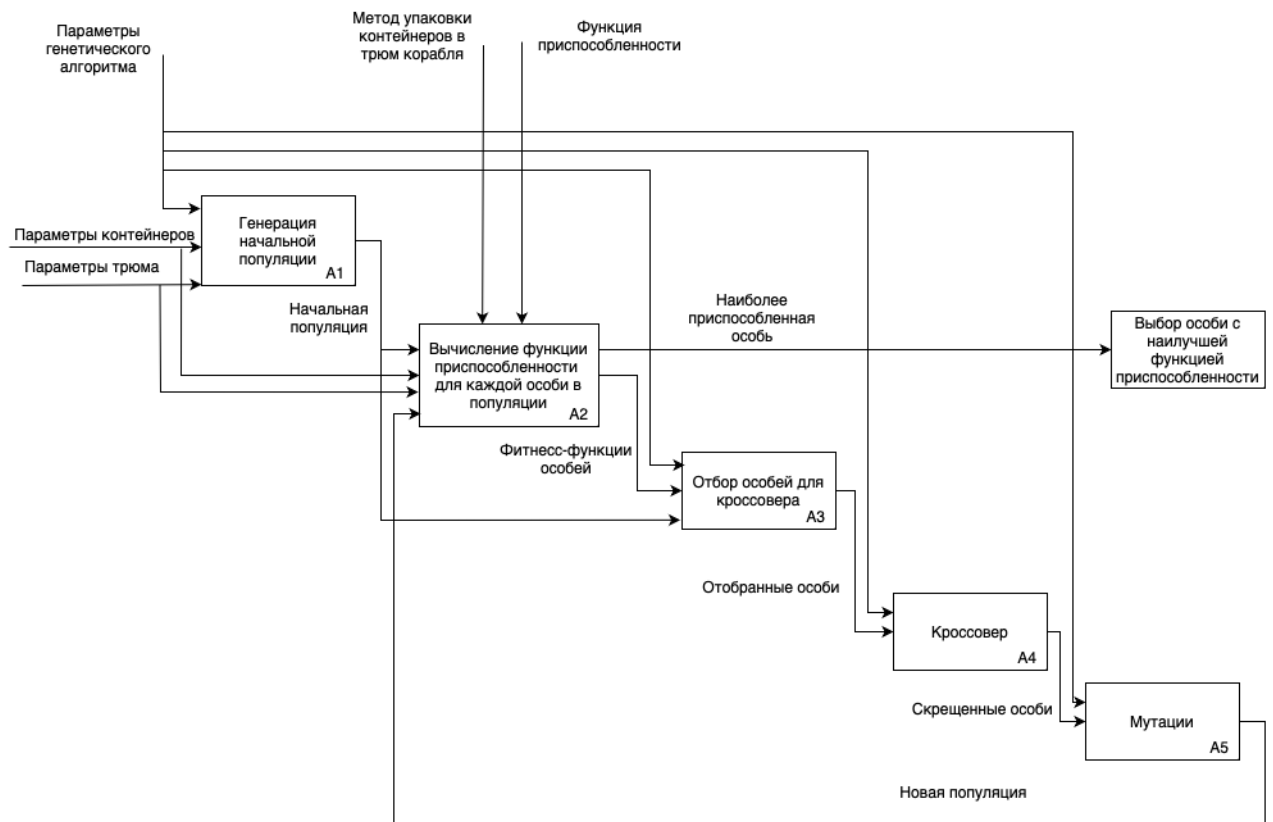


Рисунок 1 – Функциональная модель IDEF0 первого уровня для разрабатываемого метода

### 2.1.2 Схема генетического алгоритма

На рисунке 2 представлена общая схема работы генетического алгоритма [10], где Hold - параметры генетического алгоритма, Containers - массив с параметрами контейнеров, N - размер популяции, G - количество поколений, M - процент мутаций, E - процент элитных особей в популяции, T - количество особей в турнире.

Генетический алгоритм должен возвращать особь с наилучшим значением функции приспособленности.

Наилучшая особь должна содержать информацию о координатах, суммарном объеме и весе упакованных контейнеров , а также массив идентификаторов неупакованных контейнеров.



Рисунок 2 – Общая схема генетического алгоритма

### 2.1.3 Кодирование особей

Для кодирования особей было решено выбрать порядок упаковки контейнеров в трюм корабля [12]. Такой подход обеспечивает простоту операторов кроссовера и мутации, так как работа ведется с линейной перестановкой в массиве, а также сокращается время генерации начальной популяции. Данное кодирование особей позволяет переложить проверку выполнения ограничений (отсутствие пересечений контейнеров) на внутренний алгоритм упаковки, который последовательно укладывает контейнеры согласно хромосоме, гарантируя корректность получаемой особи. Таким образом, хромосома представляет собой массив идентификаторов контейнеров, определяющую порядок их упаковки в трюм (Рисунок 3).

5	1	3	2	4	6	8	10	9	7
---	---	---	---	---	---	---	----	---	---

Рисунок 3 – Представление хромосомы

Можно было бы представить особь в виде координат самого ближнего и самого дальнего углов от начала координат, однако данный подход был отвергнут по нескольким причинам. Во-первых, он значительно увеличивает пространство решений, что затрудняет работу генетического алгоритма. Во-вторых, большинство сгенерированных, скрещенных или модифицированных особей будут некорректными из-за пересечений контейнеров, и вычисление фитнес-функции становится более трудоемким, так как требует проверки всех ограничений для каждой особи.

### 2.1.4 Генерация начальной популяции

На рисунке 4 представлена схема генерации начальной популяции в генетическом алгоритме, где  $N$  - количество особей в популяции, а  $IDs$  - массив идентификаторов всех контейнеров.

Начальная популяция формируется путем генерации случайной перестановки идентификаторов контейнеров для каждой новой особи. Для обеспечения разнообразия решений, в процессе генерации популяции выполняется проверка на повторение одинаковых особей: перед добавлением новой особи в популяцию осуществляется сравнение с уже существующими, и только в случае ее уникальности она добавляется в популяцию. Такой подход исключает наличие идентичных решений, что способствует более эффективному

исследованию пространства решений генетическим алгоритмом.

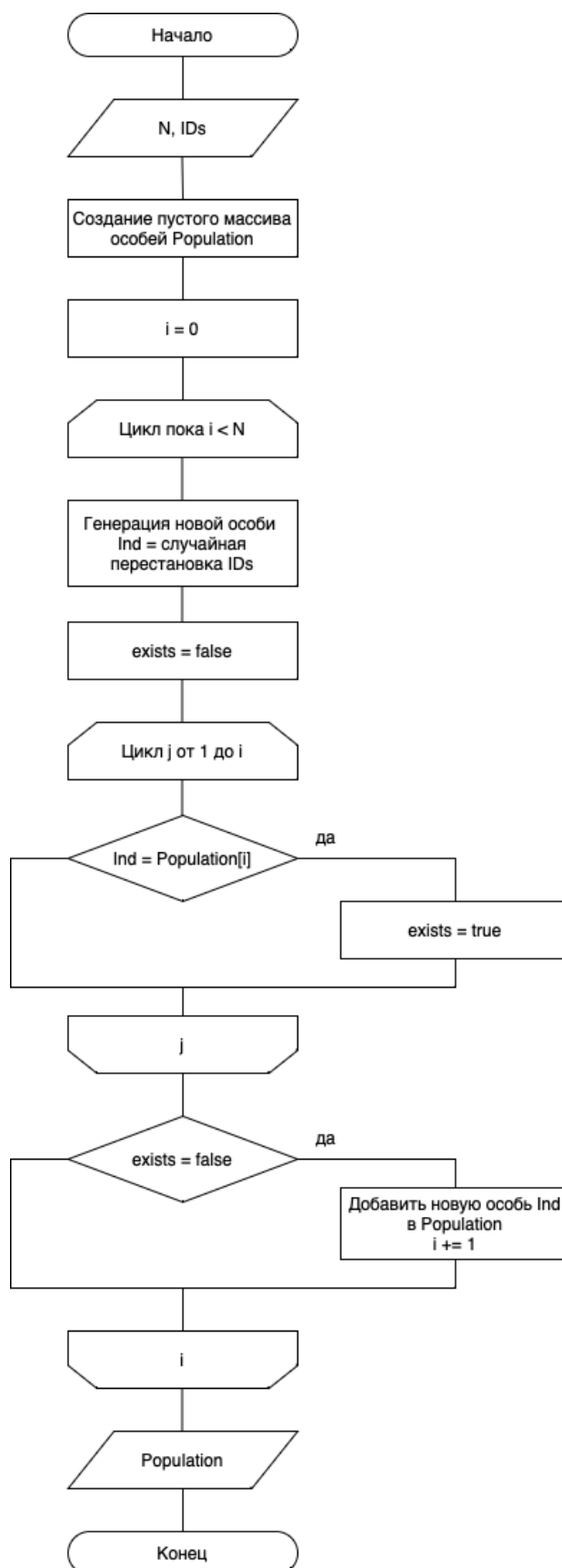


Рисунок 4 – Схема генерации начальной популяции

### 2.1.5 Вычисление фитнес-функции

Функция приспособленности в данном методе равна объему оставшегося пустого пространства в трюме после упаковки всех возможных контейнеров.

$$fitness = V_{hold} - \sum_{i=1}^P V_i \rightarrow min, \quad (2.1)$$

где  $V_{hold}$  – объем трюма,  $V_i$  – объем упакованного контейнера,  $P$  – количество упакованных контейнеров.

Хромосомы особей представляют собой порядок упаковки, но этот порядок сам по себе не гарантирует, что все контейнеры удастся разместить — часть из них может не войти из-за ограничения грузоподъемности или нехватки свободного места. Так как вычисление функции приспособленности напрямую зависит от реального размещения контейнеров в трюме, поэтому для каждой особи необходимо выполнить полную процедуру упаковки.

В разрабатываемом методе решено реализовать алгоритм упаковки контейнеров в трюм корабля на основе подхода Empty Maximal Spaces (EMS) [13]. Основная идея заключается в динамическом отслеживании максимально возможных пустых пространств в трюме после размещения каждого контейнера. Для упрощения математических вычислений один из углов трюма принимается за начало координат.

Изначально все пространство трюма рассматривается как одно большое EMS, охватывающее весь доступный объем от точки  $(0, 0, 0)$  до границ трюма. Процесс упаковки начинается с последовательной обработки контейнеров в порядке, заданном хромосомой особи. Для каждого контейнера сначала проверяется, не приведет ли его добавление к превышению грузоподъемности трюма. Затем алгоритм ищет наименьшее подходящее EMS, в которое может поместиться текущий контейнер. Алгоритм размещает контейнер в том углу EMS, который расположен ближе всего к началу координат. При этом рассматриваются две возможные ориентации контейнера – изначальная и повернутая по горизонтали, что увеличивает вероятность успешного размещения. Критерием выбора одной из двух ориентаций является максимальное удаление от дальнего угла трюма.

Рисунок 5 демонстрирует схему размещения контейнера в ems. Алгоритм возвращает координаты упакованного контейнера, если удалось его разместить в пространстве. Здесь ems - массив текущих EMS.

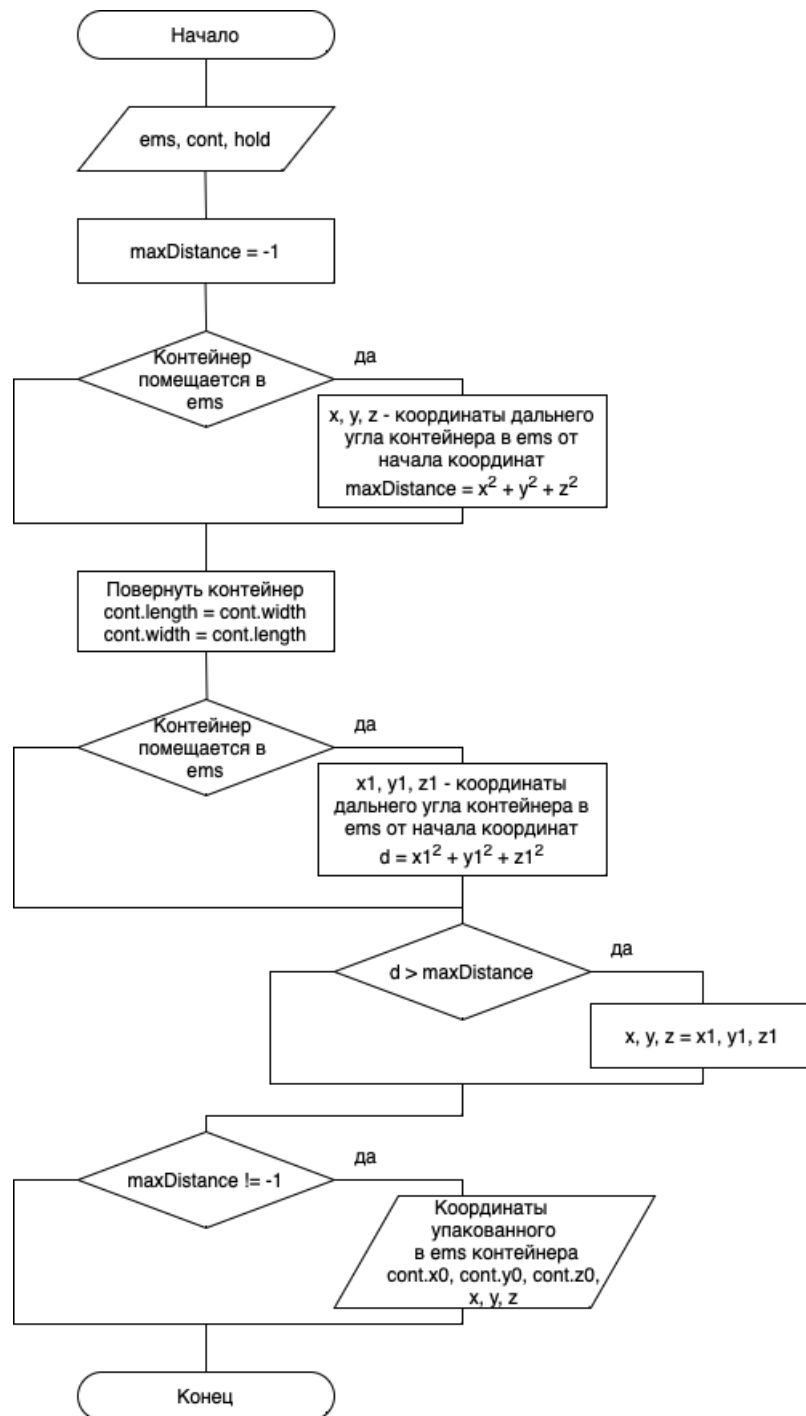


Рисунок 5 – Схема размещения контейнера в ems



Рисунок 6 демонстрирует, как размещение первого контейнера делит изначальный EMS на три новых. Также образуются еще 3 новых EMS - слева, снизу и сзади от контейнера, однако они некорректны, так как являются плоскостями, и в них нельзя ничего разместить. Таким образом, после размещения контейнера старый EMS удаляется, и добавляются 3 новых.

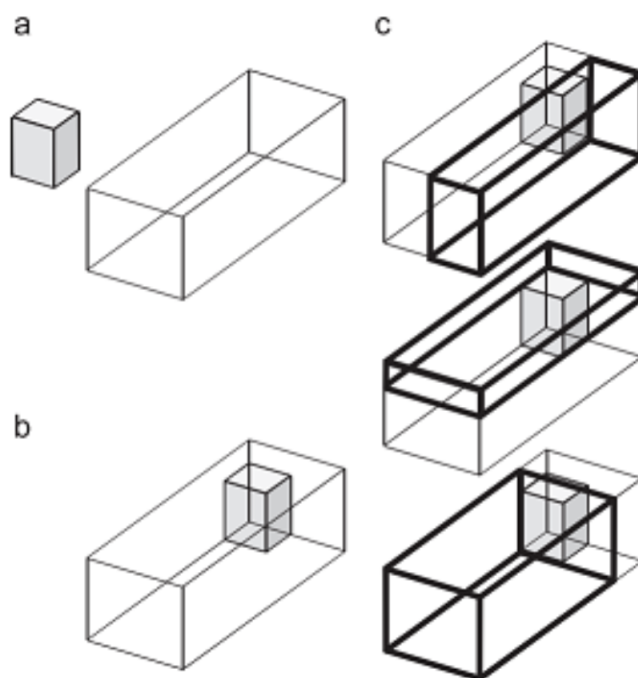


Рисунок 6 – Образование новых ems после размещения первого контейнера

После успешного размещения контейнера происходит обновление списка EMS. Текущее EMS, в котором был размещен контейнер, удаляется из списка и заменяется новыми EMS, которые образуются вокруг только что размещенного контейнера. Эти новые пространства проходят проверку на валидность (пространство не может быть плоским) и объединяются в случае, если одно пространство полностью покрывает другое. Так как EMS могут пересекаться, и часть контейнера может быть размещена в соседнем пространстве, алгоритм также обновляет и их по такому же принципу. Затем все EMS сортируются по размеру для оптимизации выбора наименьшего подходящего на следующем шаге упаковки.

Рисунок 7 демонстрирует схему обновления списка EMS после размещения очередного контейнера.

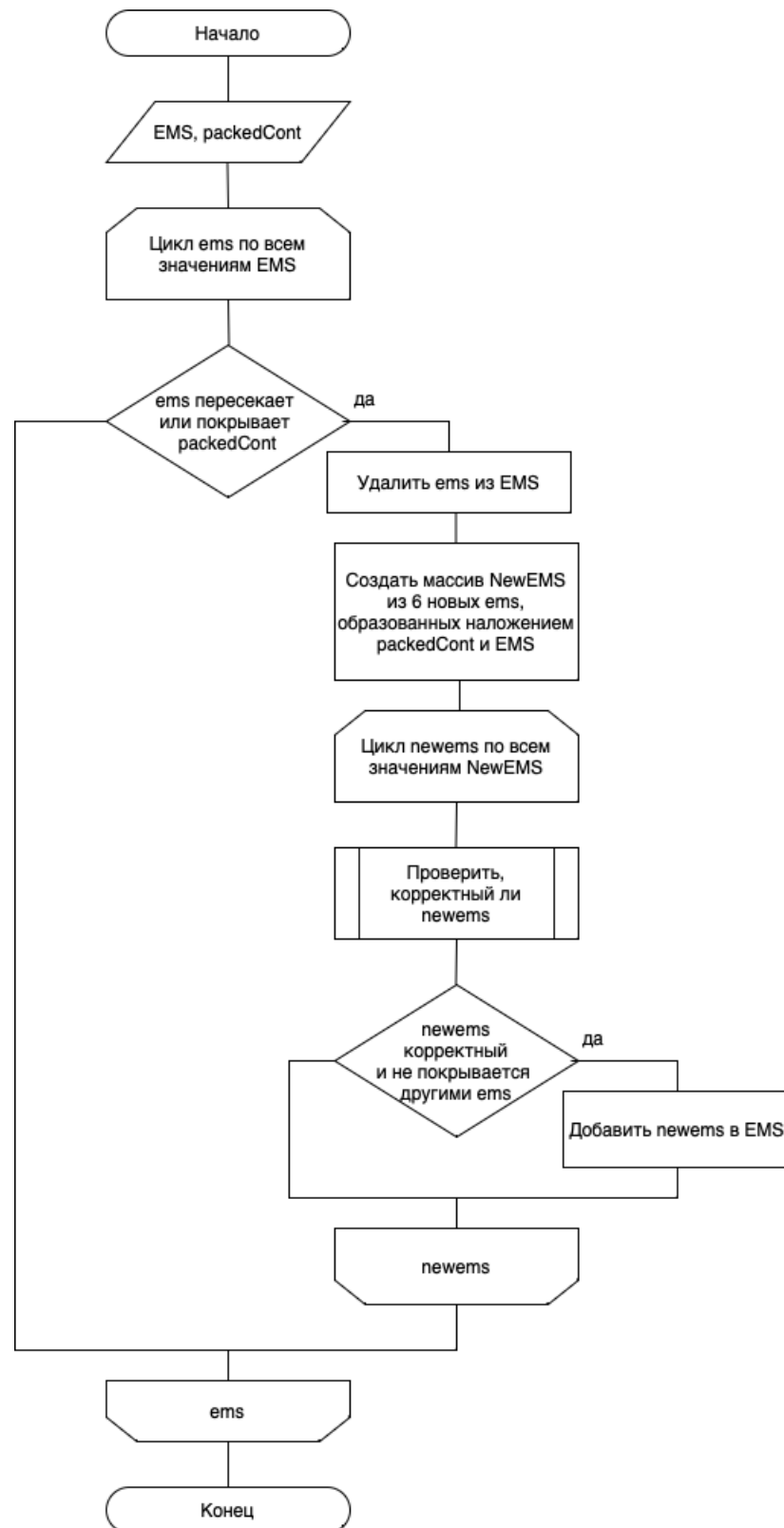


Рисунок 7 – Схема обновления списка EMS после размещения контейнера

На рисунке 8 изображена общая схема алгоритма упаковки контейнеров в трюм корабля с использованием вышеизложенного подхода. Результатом работы алгоритма являются массив с координатами упакованных контейнеров, а также их суммарный объем и вес.

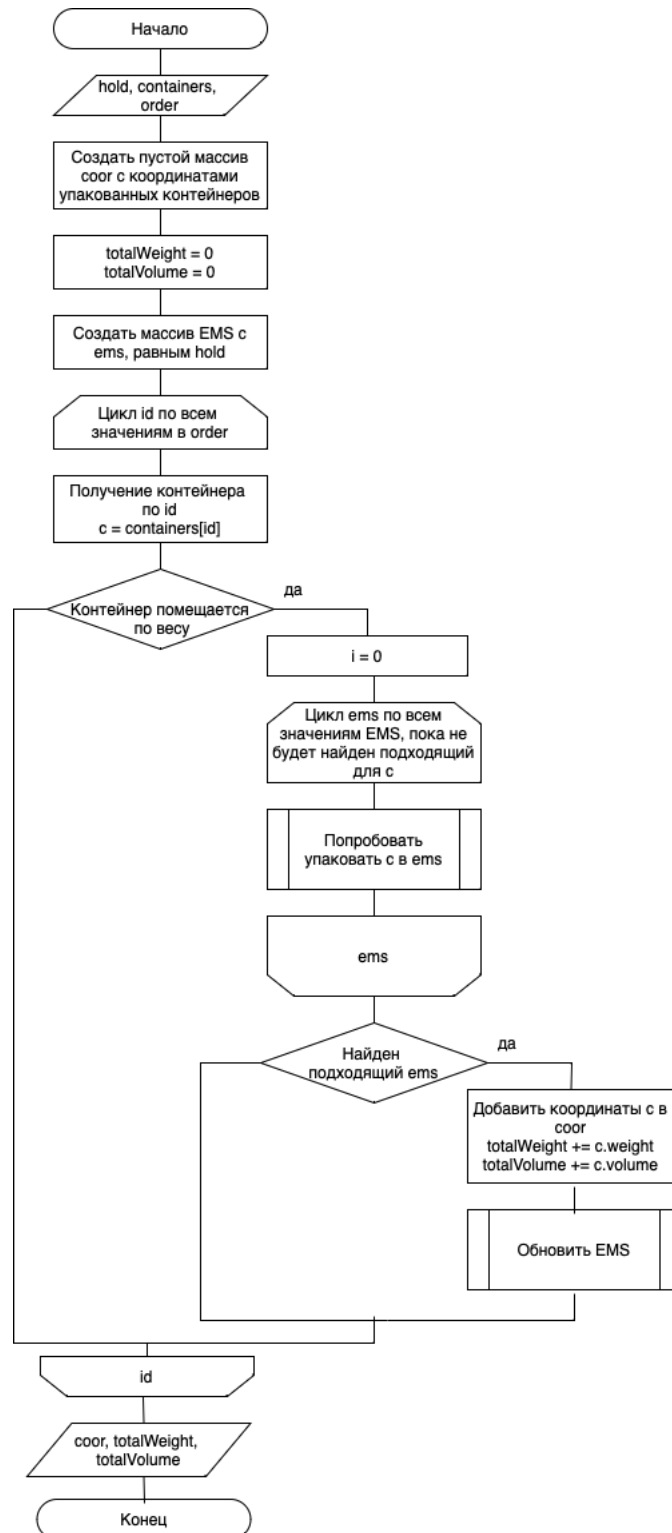


Рисунок 8 – Общая схема алгоритма упаковки контейнеров в трюм

### **2.1.6 Отбор особей**

Отбор особей в разрабатываемом методе сочетает два подхода - элитизм и турнирный отбор. В каждом поколении сперва выбираются элитные особи - лучшие решения текущей популяции, которые переходят в следующее поколение без изменений. Это гарантирует то, что популяции как минимум не будут ухудшаться. Затем для формирования родительских пар применяется турнирный отбор: случайным образом выбирается небольшая подгруппа особей (обычно 2-5), из которой выбирается одна с наименьшим значением фитнес-функции, и эта особь будет участвовать в кроссовере. Турнирный отбор позволяет поддерживать разнообразие популяции и дает шанс участвовать в скрещивании особям со средней и низкой приспособленностью, потенциально содержащим полезные гены.

На рисунке 9 изображена схема отбора элитных особей и родительских особей для дальнейшего кроссовера. Здесь *population* - текущая популяция, *N* - размер популяции, *E* - процент элитных особей, и *T* - число особей в турнире. Алгоритм возвращает массив с элитными особями и массив с отобранными для кроссовера особями.

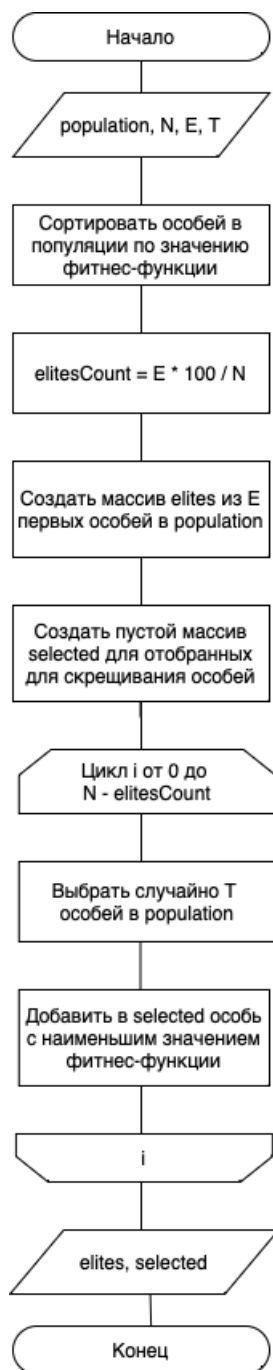


Рисунок 9 – Схема отбора в генетическом алгоритме

### 2.1.7 Кроссовер

Реализация кроссовера в разрабатываемом алгоритме основана на применении упорядоченного кроссовера, адаптированного для задачи упаковки контейнеров, где хромосома представляет собой перестановку идентификаторов контейнеров. На первом этапе выбираются две родительские особи из отобранной турнирным методом родительской группы. Затем в хромосоме первого родителя случайно выделяется сегмент генов, который напрямую переносится в потомка на те же позиции. Остальные позиции заполняются генами второго родителя в порядке их следования, исключая уже имеющиеся у потомка номера контейнеров. Аналогичная операция выполняется для второго потомка с теми же индексами сегмента, но с изменением ролей родителей. Такой подход гарантирует, что сохранятся все `id` контейнеров в хромосоме без их дублирования.

На рисунке 10 изображена схема скрещивания родительских особей и получение потомства.

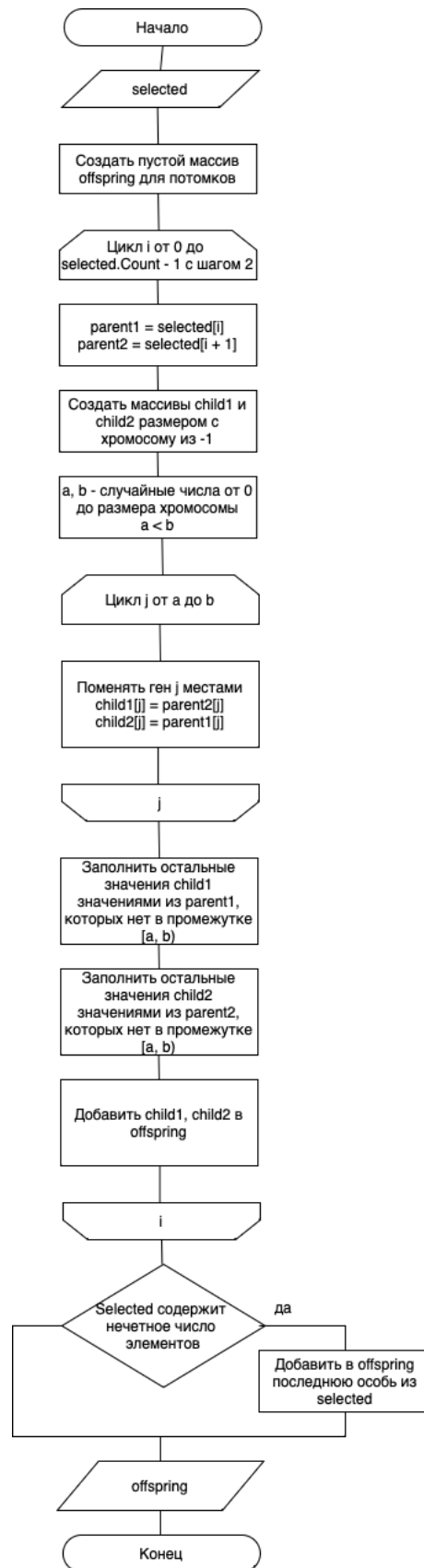


Рисунок 10 – Схема кроссовера

### 2.1.8 Мутация

Оператор мутации в генетическом алгоритме реализован как механизм случайного изменения особей, направленный на поддержание генетического разнообразия популяции. Элитные особи не подвергаются мутации и переходят в следующее поколение без изменений. К остальным особям в популяции мутация применяется с вероятностью, определяемой входным параметром генетического алгоритма. Алгоритм мутации заключается в выборе двух случайных позиций  $a$  и  $b$  в хромосоме, после чего между этими позициями обменивается подпоследовательность генов длиной от 1 до минимального расстояния между  $a$ ,  $b$  и границами хромосомы. Таким образом, вместо точечной замены одного гена мутирует целый сегмент, что повышает эффективность исследования пространства решений.



На рисунке 11 изображена схема мутации потомства. Здесь offspring - массив хромосом особей, получившихся в результате кроссовера,  $N$  - число этих особей,  $M$  - вероятность мутации.

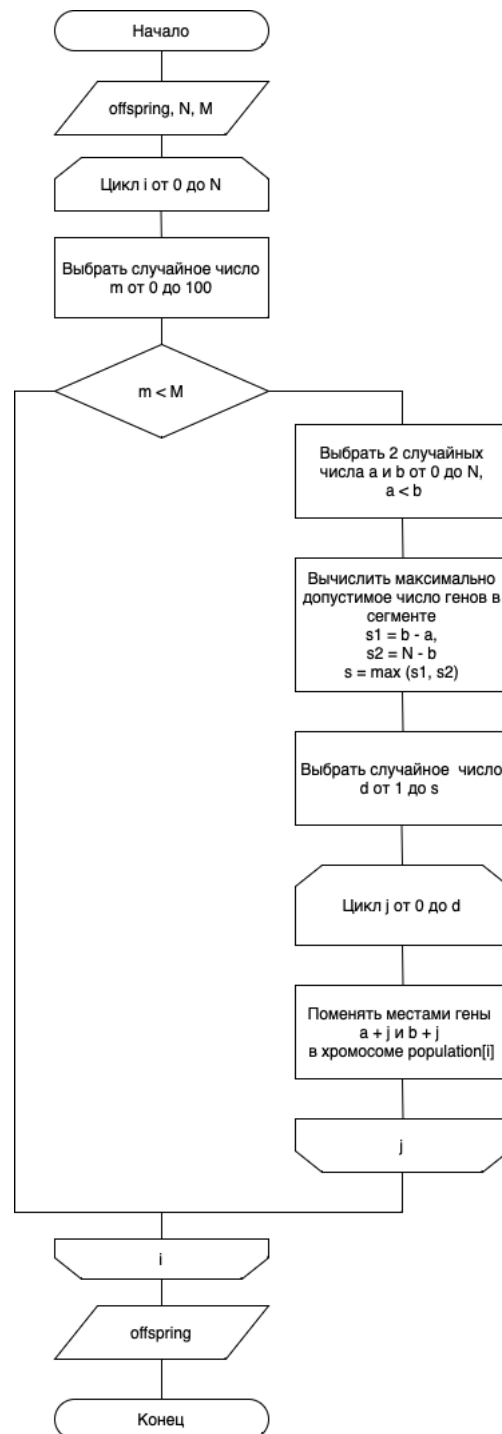


Рисунок 11 – Схема мутации

## 2.2 Разработка архитектуры программного обеспечения

Разрабатываемое программное обеспечение представляет собой десктопное приложение, написанное в объектно-ориентированном подходе. Бизнес-логика приложения включает в себя класс GeneticAlgorithm, в котором реализовывается генетический алгоритм. Работа алгоритма зависит от интерфейса IPacker, и класс PackerEMS, выполняющий алгоритм упаковки контейнеров в трюм корабля в заданном порядке, реализует данный интерфейс. Таким образом, данная инверсия зависимостей позволяет подключать различные реализации алгоритмов упаковки без изменения основного кода.

Бизнес-логика приложения содержит следующие сущности:

- ShipHold описывает характеристики трюма - грузоподъемность и габариты;
- Container описывает свойство конкретного контейнера (идентификатор, габариты, вес) и возможность поворота по горизонтали;
- PackedContainer описывает координаты упакованного контейнера;
- PackedResult хранит информацию о результате упаковки конкретной перестановки контейнеров - координаты упакованных контейнеров, их суммарный объем и вес, а также массив неупакованных контейнеров из-за ограниченного объема и грузоподъемности;
- Individual реализует концепцию особи в генетическом алгоритме, храня хромосому и значение функции приспособленности.

Для работы упаковщика PackerEMS необходимо реализовать класс EMS, хранящий информацию о координатах пространства и его объеме, а также проверку на валидность, перекрытие или пересечение с другим EMS или упакованным контейнером.

На рисунке 12 изображена UML диаграмма классов будущего приложения без учета графического интерфейса.

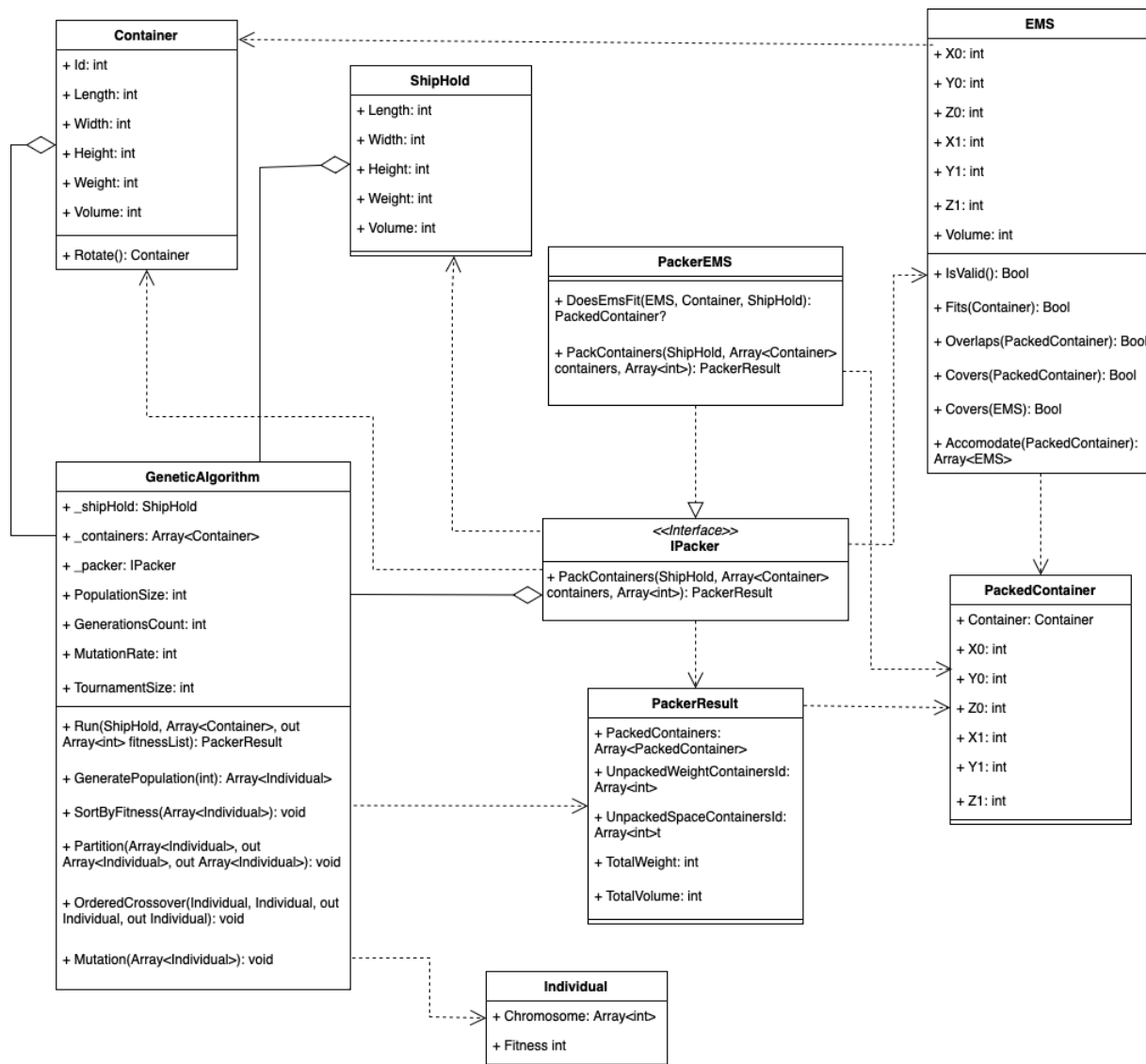


Рисунок 12 – UML диаграмма классов разрабатываемого приложения

## 2.3 Вывод

В данном разделе было описано решение задачи упаковки контейнеров в трюм корабля на основе генетического алгоритма. Модифицированы под условия задачи все шаги генетического алгоритма - генерация начальной популяции, вычисление функции приспособленности, комбинированный отбор, упорядоченный кроссовер, мутация, и описаны с помощью схем алгоритмов. Для корректного вычисления значения фитнес-функции был разработан алгоритм упаковки контейнеров в трюм корабля в заданном порядке на основе максимально возможных пустых пространств. Была разработана архитектура будущей программы и описана в виде UML диаграммы классов.

### 3 Технологический раздел

#### 3.1 Выбор средств программной реализации

Для реализации спроектированного программного обеспечения было решено использовать C# в качестве языка программирования и платформу .NET Core по нескольким причинам:

- C# — компилируемый язык программирования с JIT-оптимизацией и обеспечивают высокую производительность, что критично для вычислительно сложных задач, а поддержка многопоточности ускоряет расчет фитнес-функций [14];
- C# поддерживает ООП;
- встроенные коллекции (List<T>, Dictionary) и LINQ для работы с данными упрощают реализацию алгоритмов;
- платформа .NET Core поддерживает кроссплатформенность и позволяет запускать приложение на Windows, Linux, macOS.

Для реализации графического интерфейса пользователя в качестве фреймворка была выбрана Avalonia по следующим причинам:

- Avalonia использует единую кодовую базу для рендеринга UI на Windows, Linux, macOS [15];
- аппаратное ускорение графики через Skia обеспечивает высокую скорость рендеринга;
- XAML-подобный синтаксис и поддержка MVVM-паттерна упрощают проектирование и разработку интерфейсов.

#### 3.2 Формат входных и выходных данных

Для работы программы необходимо ввести параметры генетического алгоритма, параметры тьюма и список параметров контейнеров.

Следующие переменные определяют работу генетического алгоритма и вводятся через графический интерфейс пользователя:

- PopulationSize — положительное целое число, определяет количество особей в популяции;
- GenerationsCount - положительное целое число, описывает количество поколений;
- MutationRate - целое число от 0 до 100, которое определяет, с какой вероятностью случается мутация особей;
- TournamentSize - целое число от 2 до 5, описывает количество особей, участвующих в турнире;
- Elitism - целое число от 0 до 100, которое определяет процент элитных особей в популяции.

Парметры трюма также вводятся через интерфейс пользователя и описываются моделью, представленной в листинге 1. Все параметры описываются как целые числа больше 0.

Листинг 1 – Реализация модели ShipHold

```

1 public class ShipHold
2 {
3     public int Length { get; }
4     public int Width { get; }
5     public int Height { get; }
6     public int LiftCapacity { get; }
7     public int Volume => Length * Width * Height;
8
9
10    public ShipHold(int length, int width, int height, int
    liftCapacity)
11    {
12        Length = length;
13        Width = width;
14        Height = height;
15        LiftCapacity = liftCapacity;
16    }
17 }
```

Контейнер в программе описывается структурой, представленной в листинге 2. Идентификатор, габариты и грузоподъемность описываются целыми положительными числами, id должен быть уникальным в списке контейнеров.

## Листинг 2 – Реализация модели Container

```
1 public class Container
2 {
3     public int Id { get; }
4     public int Length { get; }
5     public int Width { get; }
6     public int Height { get; }
7     public int Weight { get; }
8     public int Volume => Length * Width * Height;
9
10    public Container(int id, int length, int width, int height, int
    weight)
11    {
12        Id = id;
13        Length = length;
14        Width = width;
15        Height = height;
16        Weight = weight;
17    }
18
19    public Container rotate()
20    {
21        return new Container(Id, Width, Length, Height, Weight);
22    }
23 }
```

Пользователь может ввести список контейнеров вручную через графический интерфейс или с помощью csv-файла. Формат входного файла был выбран из-за удобства представления параметров контейнеров в виде таблицы и возможности экспортирования данных в .csv современными СУБД. Рисунок 13 демонстрирует, как должен выглядеть файл со списком контейнеров, наличие шапки и порядок столбцов важны. Если в файле есть несколько контейнеров с одинаковым id, только первый из них будет добавлен в список контейнеров.

id	length	width	height	weight
1	16	11	14	2
2	13	47	33	5
3	64	36	44	6
4	36	28	30	7
5	23	17	28	4

Рисунок 13 – Пример входного файла с параметрами контейнеров

Формат выходных данных представлен в листинге 3. PackedContainers описывает координаты упакованных контейнеров, TotalWeight и TotalVolume - их суммарный вес и объем соответственно, и два списка идентификаторов неразмещенных контейнеров - из-за ограниченного объема или веса.

Листинг 3 – Реализация модели PackerResult

```
1 public class PackerResult
2 {
3     public List<PackedContainer> PackedContainers { get; }
4     public List<int> UnpackedWeightContainersId { get; }
5     public List<int> UnpackedSpaceContainersId { get; }
6     public int TotalWeight { get; }
7     public int TotalVolume { get; }
8
9     public PackerResult(List<PackedContainer> packedContainers, List<
    int> unpackedWeightContainersId, List<int> unpackedSpaceContainersId,
10         int totalWeight, int totalVolume)
11     {
12         PackedContainers = packedContainers;
13         UnpackedWeightContainersId = unpackedWeightContainersId;
14         UnpackedSpaceContainersId = unpackedSpaceContainersId;
15         TotalWeight = totalWeight;
16         TotalVolume = totalVolume;
17     }
18 }
```

Генетический алгоритм также возвращает массив, содержащий информацию о лучшем значении фитнес-функции в каждом поколении, для анализа эффективности его работы.

### 3.3 Реализация алгоритма упаковки контейнеров в трюм в заданном порядке

В листинге 4 представлена реализация функции DoesEmsFit класса PackerEMS, которая проверяет, можно ли разместить контейнер в заданном пространстве EMS, выбирает ориентацию размещения контейнера и возвращает координаты упакованного контейнера PackedContainer, или null.

Листинг 4 – Реализация функции DoesEmsFit класса PackerEMS

```
1 private PackedContainer? DoesEmsFit(EMS ems, Container container, ShipHold
    shipHold)
2     {
3         int maxDistance = -1;
4         var rotations = new List<Container>() { container, container.
    rotate() };
5     }
```

```

5         PackedContainer? packedContainer = null;
6         foreach (var cont in rotations)
7         {
8             if (ems.Fits(cont))
9             {
10                 var dist_x = shipHold.Length - ems.X0 - cont.Length;
11                 var dist_y = shipHold.Width - ems.Y0 - cont.Width;
12                 var dist_z = shipHold.Height - ems.Z0 - cont.Height;
13                 var distance = dist_x * dist_x + dist_y * dist_y +
dist_z * dist_z;
14
15                 if (distance > maxDistance)
16                 {
17                     maxDistance = distance;
18                     packedContainer = new PackedContainer(container,
ems.X0, ems.Y0, ems.Z0,
19                     ems.X0 + cont.Length, ems.Y0 + cont.Width, ems
.Z0 + cont.Height);
20                 }
21             }
22         }
23
24         return packedContainer;
25 }

```

В листинге 5 представлена реализация функции PackContainers класса PackerEMS, которая ищет подходящий EMS для каждого контейнера, а также создает новые EMS после размещения очередного контейнера и обновляет список пространств, а затем сортирует его.

Листинг 5 – Реализация функции PackContainers класса PackerEMS

```

1 public PackerResult PackContainers(ShipHold shipHold, List<Container>
containers, List<int> order)
2 {
3     List<PackedContainer> coordinates = new List<PackedContainer>
>();
4     var unpackedWeightContainersId = new List<int>();
5     var unpackedSpaceContainersId = new List<int>();
6     List<EMS> EMSs = new List<EMS>() { new EMS(0, 0, 0, shipHold.
Length, shipHold.Width, shipHold.Height) };
7
8     int totalWeight = 0;
9     int totalVolume = 0;
10
11     foreach (var id in order)
12     {
13         var container = containers.First(c => c.Id == id);

```



```

14
15         if (totalWeight + container.Weight > shipHold.LiftCapacity
16     )
17     {
18         unpackedWeightContainersId.Add(container.Id);
19         continue;
20     }
21
22     PackedContainer? packedContainer = null;
23
24     foreach (var ems in EMSs)
25     {
26         packedContainer = DoesEmsFit(ems, container, shipHold)
27     ;
28
29         if (packedContainer != null) { break; }
30     }
31
32     if (packedContainer == null)
33     {
34         unpackedSpaceContainersId.Add(container.Id);
35         continue;
36     }
37
38     coordinates.Add(packedContainer);
39     totalWeight += container.Weight;
40     totalVolume += container.Volume;
41
42     foreach (var ems in EMSs.ToList())
43     {
44         if (ems.Overlaps(packedContainer) | ems.Covers(
45 packedContainer))
46         {
47             EMSs.Remove(ems);
48             var newEMSs = ems.Accomodate(packedContainer);
49
50             foreach (var newEms in newEMSs)
51             {
52                 var valid = newEms.IsValid();
53
54                 int i = EMSs.Count - 1;
55
56                 while (i >= 0 & valid)
57                 {
58                     if (EMSs[i].Covers(newEms)) { valid =
false; }

```

```

58             if (newEms.Covers(EMSs[i])) { EMSs.
RemoveAt(i); }
59             i -= 1;
60         }
61
62         if (valid) { EMSs.Add(newEms); }
63     }
64
65     }
66 }
67
68     EMSs.Sort(new EMSComparer());
69 }
70
71     return new PackerResult(coordinates,
unpackedWeightContainersId, unpackedSpaceContainersId, totalWeight,
totalVolume);
72 }

```

### 3.4 Реализация генетического алгоритма

В листинге 6 представлена реализация функции Run класса GeneticAlgorithm, которая запускает генетический алгоритм и ищет оптимальное решение.

Листинг 6 – Реализация функции Run класса GeneticAlgorithm

```

1 public PackerResult Run(ShipHold shipHold, List<Container> containers, out
List<int> fitnessList)
2     {
3         _shipHold = shipHold;
4         _containers = containers;
5
6         var population = GeneratePopulation(PopulationSize);
7         fitnessList = new List<int>();
8
9         for (int g = 0; g < GenerationsCount; g++)
10        {
11            SortByFitness(population);
12            fitnessList.Add(population[0].Fitness);
13
14            List<Individual> elites;
15            List<Individual> selected;
16            Partition(population, out elites, out selected);
17
18
19            var offspring = new List<Individual>();
20            Individual child1;

```

```

21         Individual child2;
22
23         for (int k = 0; k < selected.Count - 1; k += 2)
24         {
25             OrderedCrossover(selected[k], selected[k + 1], out
child1, out child2);
26             offspring.Add(child1);
27             offspring.Add(child2);
28         }
29
30         if (selected.Count % 2 == 1) { offspring.Add(selected[
selected.Count - 1]); }
31
32         population = elites;
33         population.AddRange(offspring);
34
35         SortByFitness(population);
36
37         Mutation(population);
38
39
40         var populationCopy = population.ToList();
41         int i = PopulationSize - 1;
42         while (i > 0)
43         {
44             int j = i - 1;
45             while (j >= 0)
46             {
47                 if (populationCopy[i].Chromosome.SequenceEqual(
populationCopy[j].Chromosome))
48                 {
49                     population.RemoveAt(i);
50                     j = 0;
51                 }
52                 j--;
53             }
54
55             i--;
56         }
57
58         population.AddRange(GeneratePopulation(PopulationSize -
population.Count));
59     }
60
61     SortByFitness(population);
62     Console.WriteLine($"end: fitness = {population[0].Fitness}");
63
64     return PackContainers(population[0]);

```

В листинге 7 представлена реализация функции генерации начальной популяции класса GeneticAlgorithm. Функция гарантирует, что все особи в популяции будут иметь уникальную перестановку идентификаторов контейнеров.

Листинг 7 – Реализация функции GeneratePopulation класса GeneticAlgorithm

```

1 private List<Individual> GeneratePopulation(int size)
2 {
3     var population = new List<Individual>();
4     var containersId = _containers.Select(c => c.Id).ToList();
5
6     while (population.Count < size)
7     {
8         var random = new Random();
9
10        var chromosome = containersId.OrderBy(x => random.Next()).
        ToList();
11
12        var exists = false;
13        for (int j = 0; !exists & j < population.Count; j++)
14        {
15            if (population[j].Chromosome.SequenceEqual(chromosome)
16        ) { exists = true; }
17
18            if (!exists)
19                population.Add(new Individual(chromosome));
20        }
21
22        return population;
23    }

```

В листинге 8 представлена реализация функции сортировки популяции по возрастанию функции приспособленности SortByFitness класса GeneticAlgorithm. Для вычисления функции приспособленности вызывается функция PackContainers.

Листинг 8 – Реализация функция SortByFitness класса GeneticAlgorithm

```

1 private void SortByFitness(List<Individual> population)
2 {
3     population.ForEach(ind => ind.Fitness = _shipHold.Volume -
        PackContainers(ind).TotalVolume);

```

```

4         population.Sort(new IndividualComparer());
5     }

```

В листинге 9 представлена реализация функции отбора элитных особей и турнирного отбора особей для скрещивания Partition класса GeneticAlgorithm.

Листинг 9 – Реализация функция Partition класса GeneticAlgorithm

```

1 private void Partition(List<Individual> sortedPopulation, out List<
    Individual> elites, out List<Individual> selected)
2 {
3     int elitesCount = Elitism * PopulationSize / 100;
4     selected = new List<Individual>();
5     elites = sortedPopulation.Take(elitesCount).ToList();
6
7     for (int i = 0; i < PopulationSize - elitesCount; i++)
8     {
9         var random = new Random();
10
11         var tournament = sortedPopulation.OrderBy(x => random.Next
            ()).Take(TournamentSize).ToList();
12         tournament.Sort(new IndividualComparer());
13         selected.Add(tournament[0]);
14     }
15 }

```

В листинге 10 представлена реализация функции скрещивания двух родительских пар и получения двух потомков OrderedCrossover класса GeneticAlgorithm. Алгоритм гарантирует, что после кроссовера гены не будут дублироваться.

Листинг 10 – Реализация функция OrderedCrossover класса GeneticAlgorithm

```

1 private void OrderedCrossover(Individual parent1, Individual parent2, out
    Individual child1, out Individual child2)
2 {
3     int size = parent1.Chromosome.Count;
4
5     var random = new Random();
6     int a = random.Next(size);
7     int b = random.Next(size);
8     if (a > b) (a, b) = (b, a);
9
10
11     var chr1 = Enumerable.Repeat(-1, size).ToList();

```

```

12         var chr2 = Enumerable.Repeat(-1, size).ToList();
13
14         for (int i = a; i < b; i++)
15         {
16             chr1[i] = parent1.Chromosome[i];
17             chr2[i] = parent2.Chromosome[i];
18         }
19
20         int index = 0;
21         for (int i = 0; i < size; i++)
22         {
23             if (i < a | i >= b)
24             {
25                 while (index < size && chr1.Contains(parent2.
Chromosome[index]))
26                     index++;
27                 chr1[i] = parent2.Chromosome[index++];
28             }
29         }
30
31         index = 0;
32         for (int i = 0; i < size; i++)
33         {
34             if (i < a || i >= b)
35             {
36                 while (index < size && chr2.Contains(parent1.
Chromosome[index]))
37                     index++;
38                 chr2[i] = parent1.Chromosome[index++];
39             }
40         }
41
42         child1 = new Individual(chr1);
43         child2 = new Individual(chr2);
44     }

```

В листинге 11 представлена реализация функции мутации особей текущего поколения Mutation класса GeneticAlgorithm, которая с заданной вероятностью меняет 2 случайных сегмента генов одинаковой длины местами.

#### Листинг 11 – Реализация функция Mutation класса GeneticAlgorithm

```

1 private void Mutation(List<Individual> population)
2 {
3     var size = _containers.Count;
4     var random = new Random();
5     for (int i = Elitism * PopulationSize / 100; i <
PopulationSize; i++)

```

```

6      {
7          if (random.Next(100) < MutationRate)
8          {
9              int a = random.Next(size);
10             int b = random.Next(size);
11             if (a > b) (a, b) = (b, a);
12             int c = size - b;
13             int d = b - a;
14             if (c > d) (c, d) = (d, c);
15
16             int genes = random.Next(c + 1);
17
18             for (int j = 0; j < genes; j++)
19                 (population[i].Chromosome[a + j], population[i].
Chromosome[b + j]) = (population[i].Chromosome[b + j], population[i].
Chromosome[a + j]);
20         }
21     }
22 }

```

### 3.5 Демонстрация работы программы

Рисунок 14 демонстрирует интерфейс, который видит пользователь при запуске приложения. Интерфейс разделен на 4 части: ввод параметров генетического алгоритма, ввод параметров трюма, ввод списка контейнеров и вывод результата работы программы. Если какие-то параметры заданы неверно, программа выдает сообщение-подсказку о формате входных данных.

Упаковка контейнеров в трюм с использованием генетического алгоритма

**Параметры генетического алгоритма**

Размер популяции:  Количество особей в турнире:

Количество поколений:  Процент элитизма (%):

Процент мутации (%):

---

**Параметры трюма**

Длина (дм):  Высота (дм):

Ширина (дм):  Грузоподъемность (т):

---

**Контейнеры**

---

Рисунок 14 – Внешний вид интерфейса при запуске программы

Рисунок 15 демонстрирует ввод списка контейнеров через csv-файл. При нажатии на кнопку "Загрузить из csv" приложение открывает менеджер файлов, в котором пользователь может выбрать нужный файл.

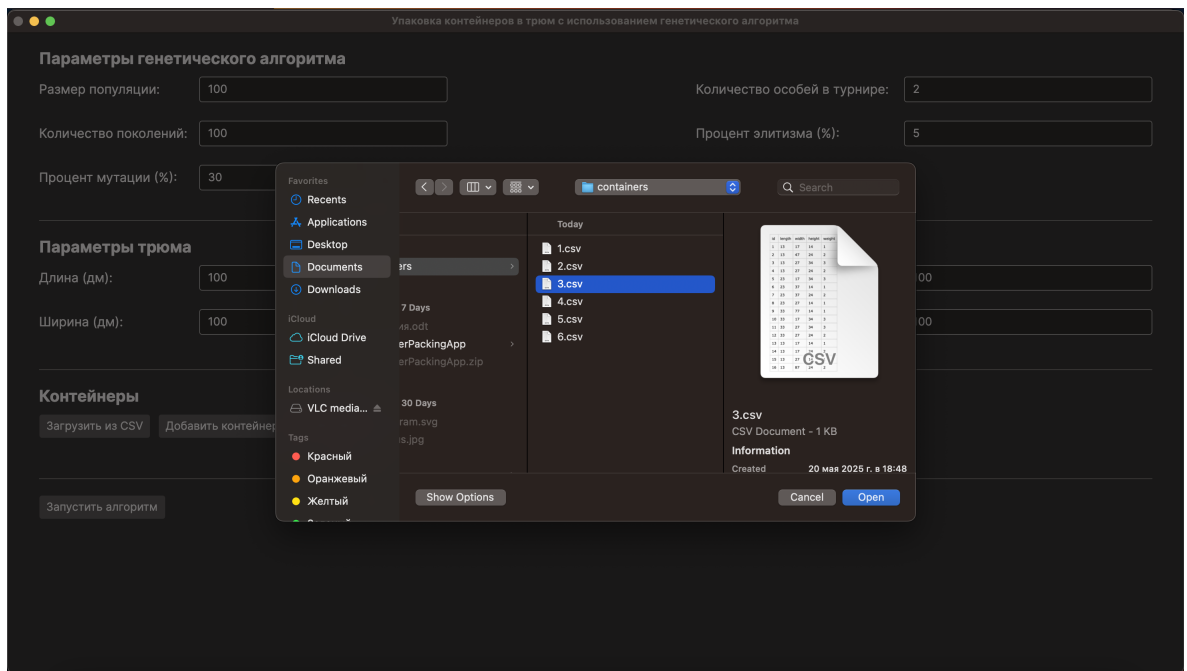


Рисунок 15 – Внешний вид окна выбора файла для ввода списка контейнеров

Рисунок 16 демонстрирует, как выглядит список контейнеров в программе. Пользователь может поменять все параметры в данном списке, кроме id.

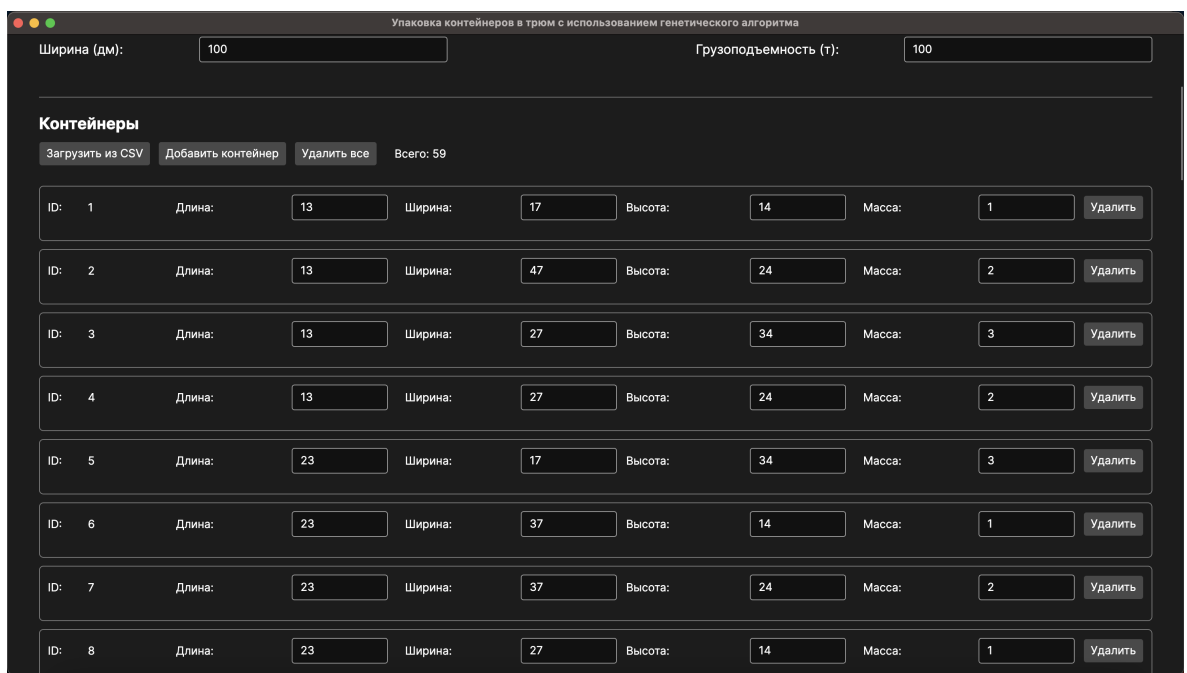


Рисунок 16 – Демонстрация интерфейса для ввода параметров контейнеров

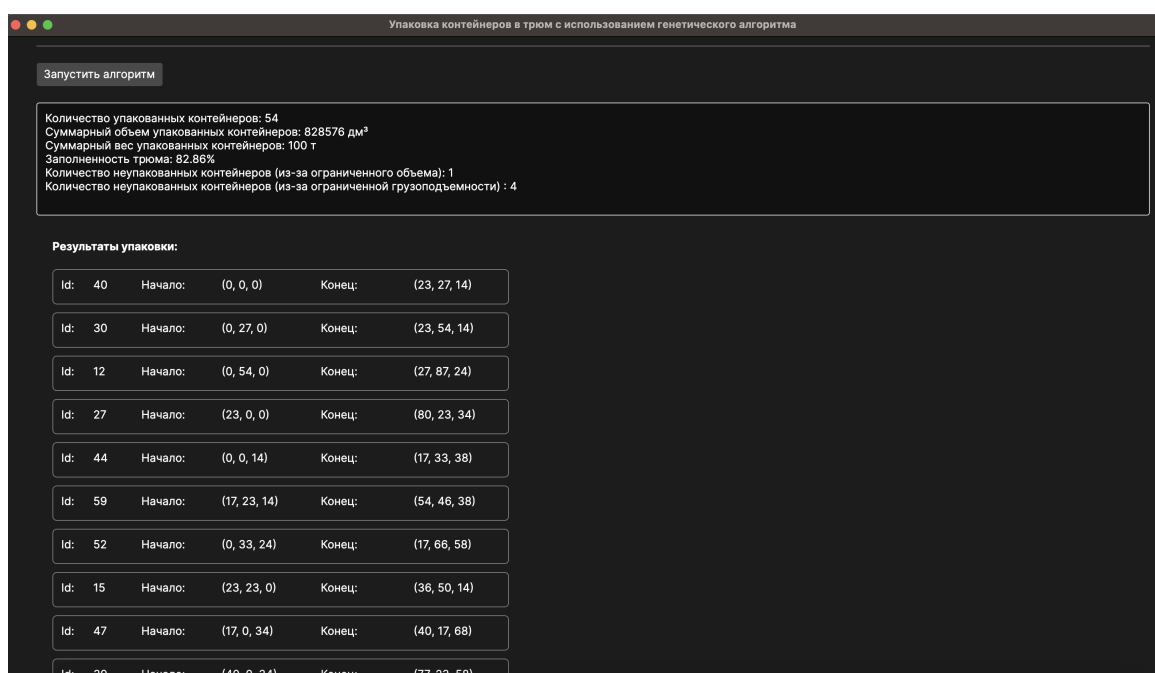


После ввода всех необходимых данных, для запуска процесса упаковки нужно нажать кнопку "Запустить алгоритм". Если какие-то данные введены неверно, программа выдаст сообщение об ошибке (рисунок 17).



Рисунок 17 – Демонстрация вывода сообщения об ошибке при вводе неверных параметров

Результат работы приложения изображен на рисунке 18.



Id	Начало	Конец
40	(0, 0, 0)	(23, 27, 14)
30	(0, 27, 0)	(23, 54, 14)
12	(0, 54, 0)	(27, 87, 24)
27	(23, 0, 0)	(80, 23, 34)
44	(0, 0, 14)	(17, 33, 38)
59	(17, 23, 14)	(54, 46, 38)
52	(0, 33, 24)	(17, 66, 58)
15	(23, 23, 0)	(36, 50, 14)
47	(17, 0, 34)	(40, 17, 68)
39	(40, 0, 34)	(77, 23, 58)

Рисунок 18 – Демонстрация результата работы программы

Для анализа работы генетического алгоритма приложение выводит график зависимости процента заполненности трюма от номера поколения (рисунок 19).

### 3.6 Вывод

В данном разделе был проведен и обоснован выбор средств программной реализации, описан формат входных и выходных данных. Приведена реализация генетического алгоритма и алгоритма упаковки контейнеров в трюм в заданном порядке. Были описаны интерфейс и взаимодействие пользователя с разработанным приложением.

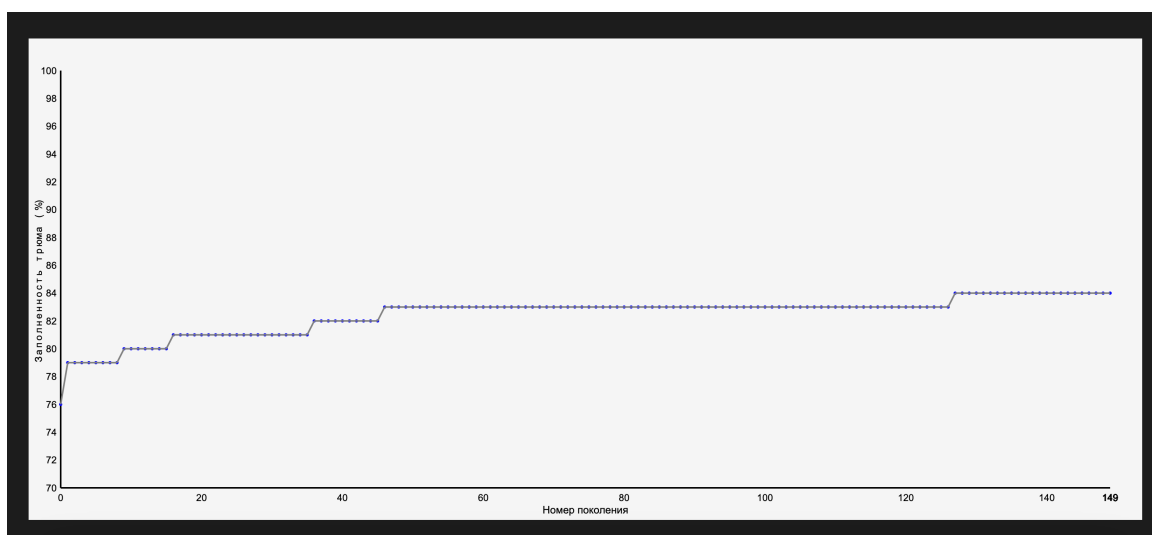


Рисунок 19 – Демонстрация результата работы программы в виде графика

## **4 Исследовательский раздел**

### **4.1 Проведение исследований**

Исследования проводились на ноутбуке со следующими техническими характеристиками:

- операционная система macOS Ventura 13.4;
- память 8 ГБайт;
- процессор Apple M2 (8-ядерный: 4 производительных и 4 энергоэффективных);
- процессор имеет 8 физических и 8 логических ядер (архитектура ARM, без Hyper-Threading).

Во время проведения исследований устройство было подключено к сети электропитания.

Во всех исследованиях для описания габаритов контейнеров были использованы реальные данные о стандартных [16] и нестандартных контейнерах [17]. Масса каждого контейнера была выбрана случайно в пределах указанной в источниках максимальной массы груза. Размеры и грузоподъемность фидерных контейнеровозов были взяты с официального интернет-магазина судов [18].

Размер трюма и грузоподъемность для каждого испытания были специально подобраны так, чтобы было невозможно упаковать все контейнеры.

### **4.2 Проверка адекватности построенной модели**

Генетический метод является итерационным и подразумевает, что с каждым поколением алгоритм стремится улучшить качество решений, отбирая более приспособленные особи и комбинируя их признаки, приближаясь к глобальному оптимуму. Назовем это сходимостью зависимости целевой функции от поколения к глобальному минимуму. Однако если глобальный минимум неизвестен из-за большого размера входных данных, характер сходимости можно оценить визуально по графику изменения целевой функции в зависимости от поколения. Адекватность модели определяется на основе полученной информации о сходимости.

Во всех испытаниях использовались значения параметров генетического метода, представленные в таблице 2:

Таблица 2 – Параметры генетического метода

Параметр	Значение
Процент мутации М	20
Процент элитных особей Е	5
Количество особей в турнире Т	2

Количество поколений и размер популяции варьируется в зависимости от числа контейнеров.

Для подтверждения адекватности построенной модели было проведено исследование сходимости зависимости функции приспособленности от поколения на небольшом и большом количестве контейнеров.

Для исследования оптимальности на небольшом количестве контейнеров возьмем 9 стандартных контейнеров (4 типа 20-футовых, 4 типа 40-футовых и один 30-футовый) и сравним значение целевой функции с глобальным оптимумом, полученным с помощью метода полного перебора. Такое количество контейнеров является макисмальным для испоьнования метода полного перебора, так как при увеличении их числа время работы резко возрастает. Это объясняется тем, что количество всех перестановок равно  $n!$ , где  $n$  - количество контейнеров. А так как у каждого контейнера есть две возможные ориентации в трюме, то количество итераций в методе полного перебора увеличивается до  $n! * 2^n$ .

График сходимости зависимости целевой функции от поколения к глобальному минимуму, найденному с помощью метода полного перебора, представлен на рисунке 20. Для наглядности исследования размер популяции N был уменьшен до 10, а количество поколений G - до 50.

График демонстрирует, что с увеличением номера поколения значение целевой функции приближаются к глобальному минимуму, равному 65260, и достигает его в 26 поколении. Далее, из-за того что лучшие особи переходят в следующее поколение без изменений, значение фитнес-функции не меняется. Исходя из полученных результатов можно сделать вывод, что на небольшом количестве контейнеров генетический метод может получить настолько же оптимальное решение, как и метод полного перебора.

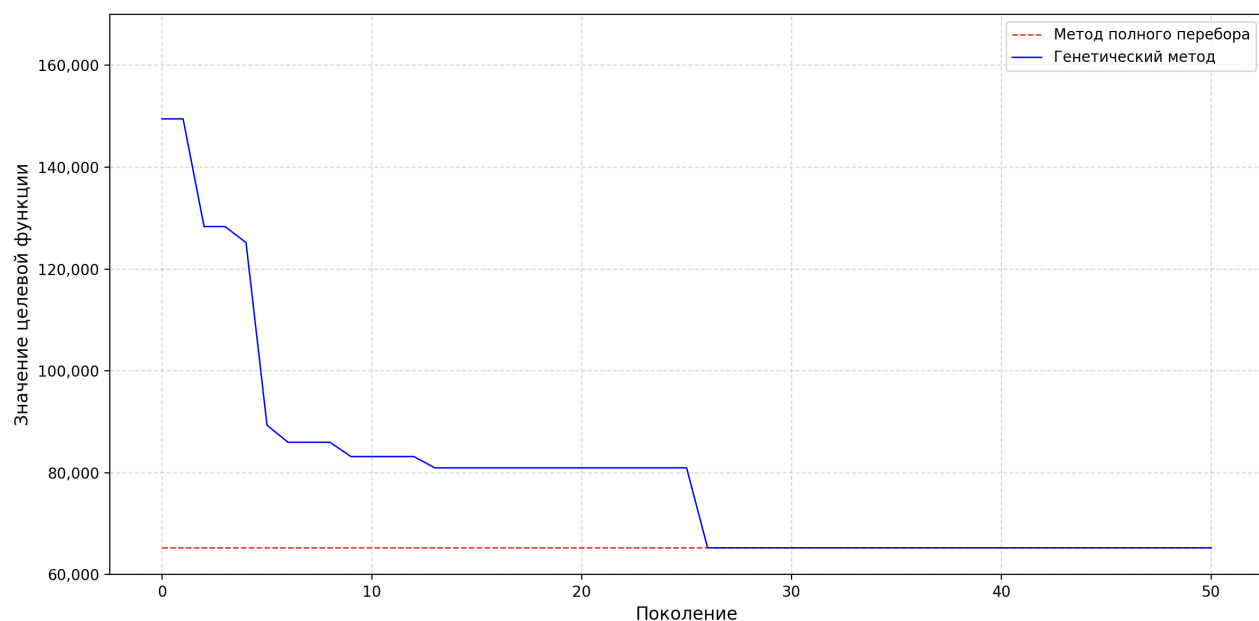


Рисунок 20 – График сходимости зависимости целевой функции от поколения к глобальному минимуму, найденному методом полного перебора

Далее оценим адекватность работы метода на большом количестве контейнеров. Проведем 3 испытания с количеством контейнеров равным 50, 100 и 200. Метод полного перебора неприменим к такому большому набору входных данных, поэтому оценим сходимость визуально по графику изменения целевой функции в зависимости от поколения.

График сходимости зависимости целевой функции от номера поколения на большом количестве контейнеров, равном 50, представлен на рисунке 21. Размер популяции  $N$  равен 100, а количество поколений  $G$  - 200.

График сходимости зависимости целевой функции от номера поколения на большом количестве контейнеров, равном 100, изображен на рисунке 22. Размер популяции  $N$  равен 200, а количество поколений  $G$  - 400.

График сходимости зависимости целевой функции от номера поколения на большом количестве контейнеров, равном 200, представлен на рисунке 23. Размер популяции  $N$  равен 400, а количество поколений  $G$  - 500.

По трем графикам видно, что с увеличением номера поколения получаемые значения целевой функции монотонно уменьшаются. Из этого можно сделать вывод, что на большом количестве контейнеров реализация генетического метода постепенно улучшает качество решения.

На основе проведенного исследования можно сделать вывод, что зависимость значений целевой функции от увеличения номера поколения реали-

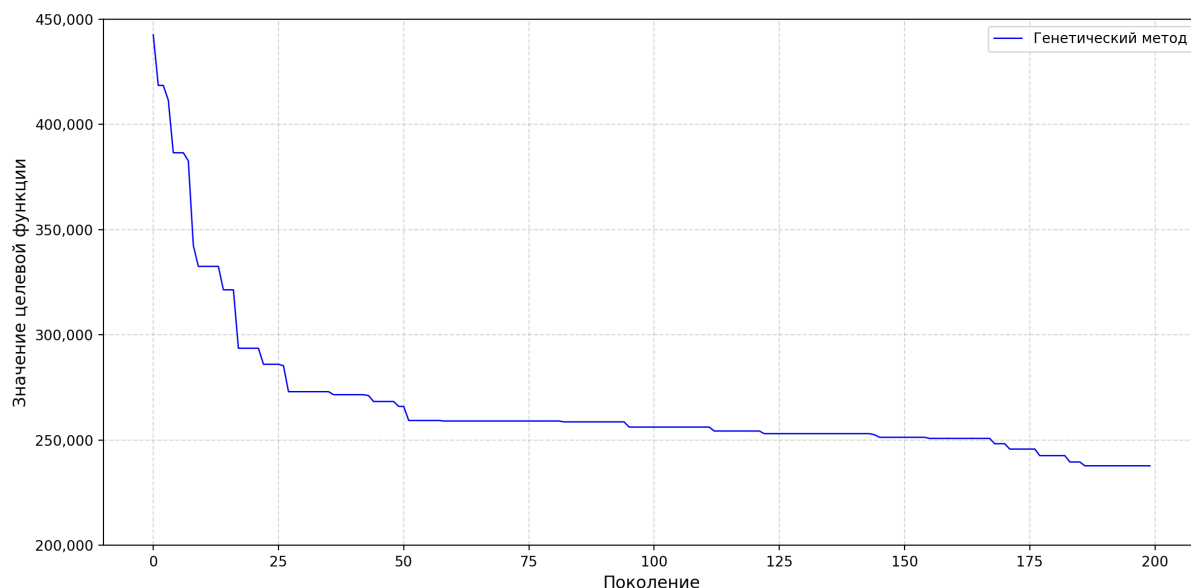


Рисунок 21 – График сходимости зависимости целевой функции от номера поколения к глобальному минимуму на большом количестве контейнеров, равном 50

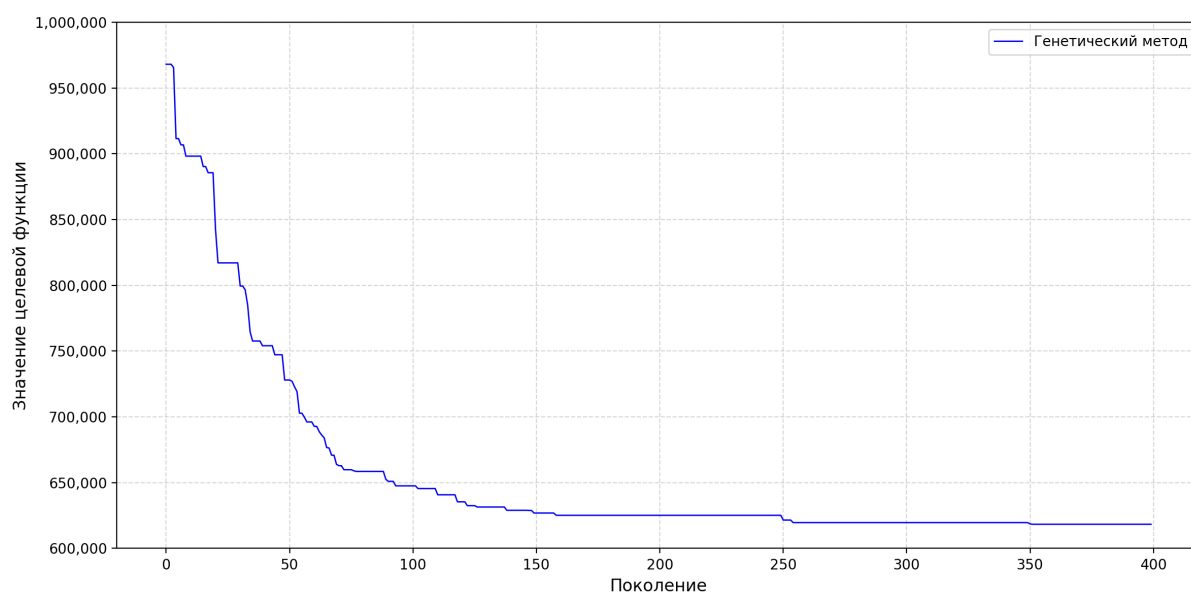


Рисунок 22 – График сходимости зависимости целевой функции от номера поколения к глобальному минимуму на большом количестве контейнеров, равном 100

зованного генетического метода стремится к некоторому глобальному минимуму. В ситуации с небольшим количеством контейнеров было показано, что генетический метод способен даже достичь глобальный минимум, полученный с помощью метода полного перебора. Результаты исследования подтверждают адекватность построенной модели.

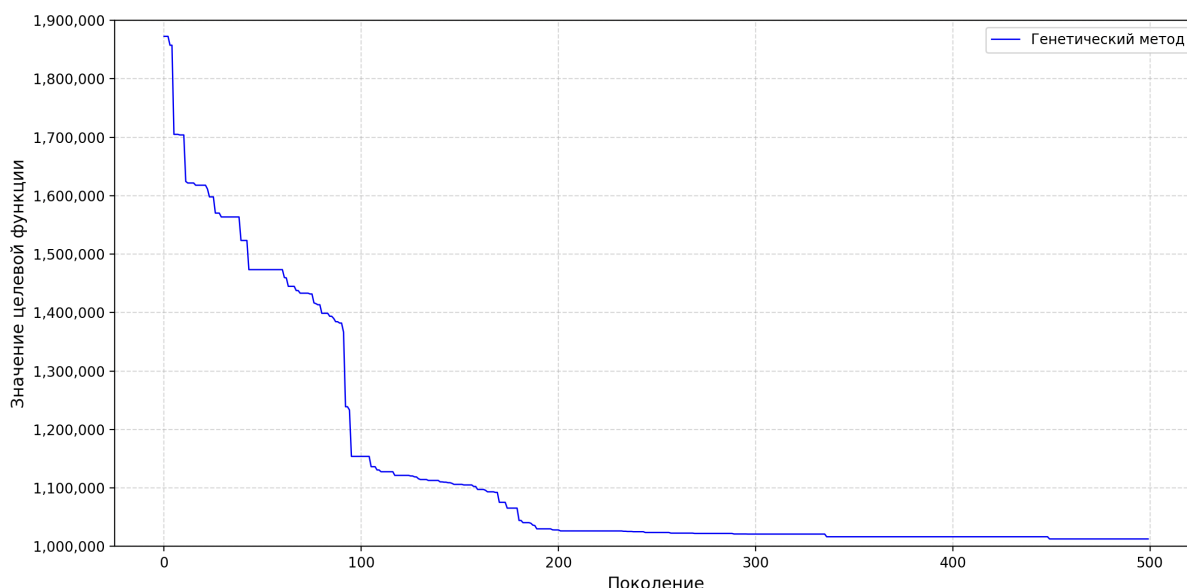


Рисунок 23 – График сходимости зависимости целевой функции от номера поколения к глобальному минимуму на большом количестве контейнеров, равном 200

#### 4.3 Исследование зависимости сходимости от процента элитизма и количества особей в турнире

Элитизм и турнирный отбор в генетическом методе находятся в сложной взаимозависимости и существенно влияют на эффективность решения задачи. Оба параметра влияют на баланс между двумя фундаментальными процессами эволюции: эксплуатацией найденных хороших решений и исследованием пространства поиска. Элитизм, сохраняя лучшие особи, усиливает эксплуатацию, в то время как турнирный отбор, в зависимости от своего размера, может как способствовать исследованию (при малых размерах турнира), так и усиливать эксплуатацию (при больших размерах).

Целью данного исследования является изучение влияния процента элитизма и количества особей в турнире друг на друга и выявление оптимальных сочетаний этих параметров для задачи упаковки контейнеров в трюм.

Исследование проводилось на массиве из 50 стандартных контейнеров, метод полного перебора для такого количества неприменим. Для каждого испытания было выполнено 30 независимых прогонов для повышения достоверности данных. Во всех испытаниях использовались значения параметров генетического метода, представленные в таблице 3:

Критерием оценки эффективности метода служит сходимость зависи-

Таблица 3 – Параметры  
генетического метода

Параметр	Значение
Размер популяции N	100
Количество поколений G	200
Вероятность мутации M	20

мости фитнес-функции от номера поколения при значениях параметров элитизма  $E = 0\%$ ,  $10\%$ ,  $20\%$ ,  $30\%$  и размера турнира  $T = 2, 3, 4$ .

График сходимости зависимости функции приспособленности от поколения при  $T = 2$  представлен на рисунке 24.

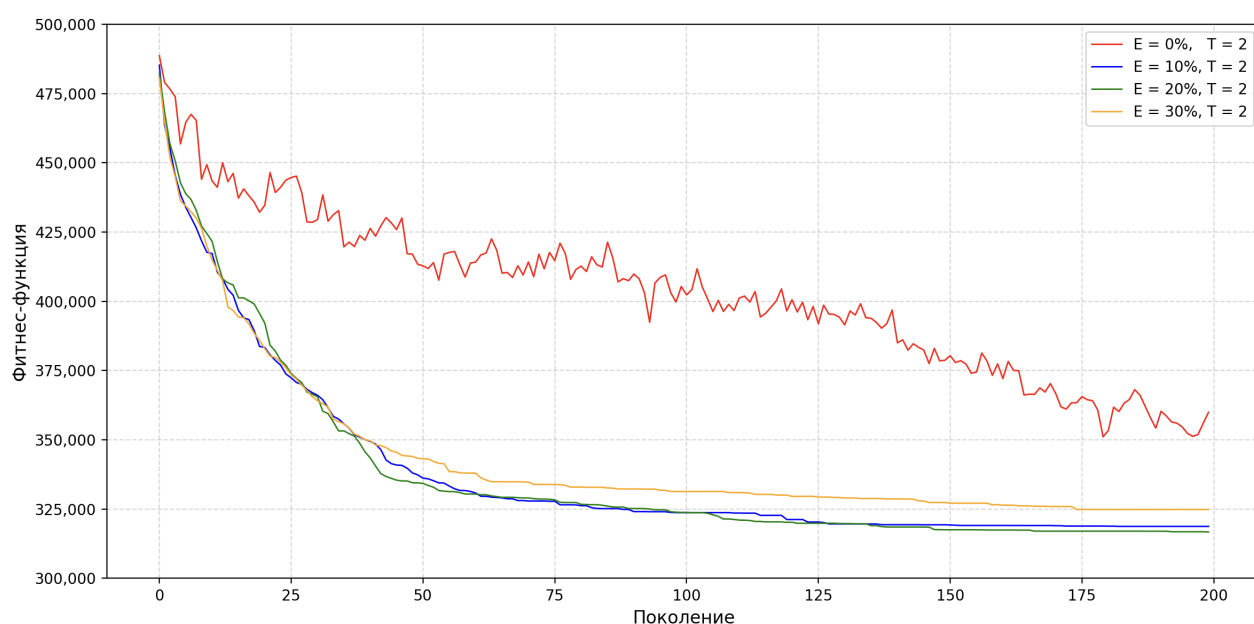


Рисунок 24 – График сходимости зависимости целевой функции от номера поколения при  $T = 2$

На графике видно, что при размере турнира  $T = 2$  отсутствие элитизма приводило к очень плохой сходимости и относительно низкому качеству конечного решения. Введение элитизма на уровне 10-20% способствовало улучшению сходимости при сохранении приемлемого уровня разнообразия популяции. Однако дальнейшее увеличение элитизма до 30% вызывало ухудшение эффективности и преждевременную сходимость к субоптимальному решению. Это означает, что при слабом турнире ( $T = 2$ ) умеренный элитизм компенсирует недостаточное давление отбора, улучшая сходимость.

Далее, было проведено аналогичное испытание при турнирном отборе  $T = 3$ . График сходимости зависимости функции приспособленности от по-



коления представлен на рисунке 25.

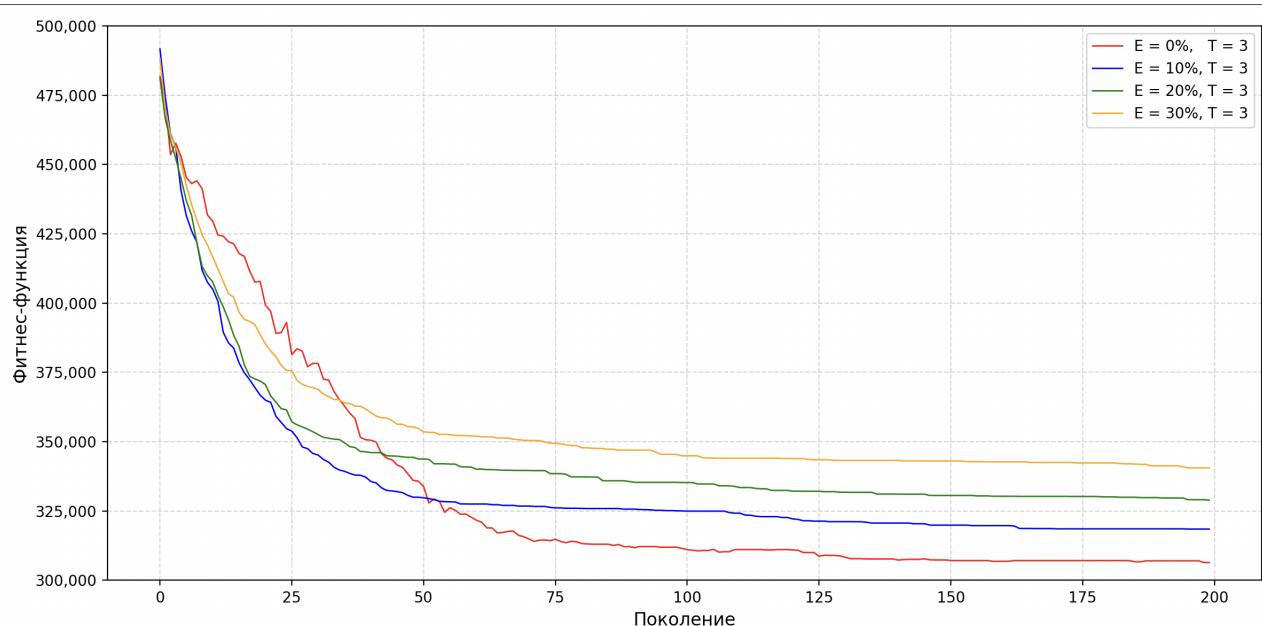


Рисунок 25 – График сходимости зависимости целевой функции от номера поколения при  $T = 3$

На графике видно, что наилучшие результаты достигались при полном отсутствии элитизма, что объясняется достаточным давлением отбора, создаваемым турниром большего размера. Добавление элитизма в этом случае приводило к ухудшению сходимости из-за застревания в локальном минимуме.

Аналогичная тенденция наблюдалась для количества особей в турнире  $T = 4$ , однако слишком сильное давление отбора даже при полном отсутствии элитизма приводит к чрезмерной эксплуатации и ухудшению качества конечного решения. График сходимости зависимости функции приспособленности от поколения при  $T = 4$  представлен на рисунке 26.

На основе результатов проведенного исследования можно сделать вывод, что при слабом турнире ( $T = 2$ ) элитизм компенсирует слабое давление отбора и улучшая сходимость. Однако при увеличении размера турнира дополнительное сохранение лучших особей через элитизм приводит к чрезмерной эксплуатации и потере разнообразия, что негативно сказывается на качестве решения. При этом, при дальнейшем увеличении размера турнира отбор становится слишком жестким, что приводит к быстрой потере генетического разнообразия в популяции. Проведенное исследование позволило установить, что оптимальной комбинацией данных параметров являются  $T$

= 2 и  $E = 10-20$  процентов, и  $T = 3$  и  $E = 0-10$  процентов.

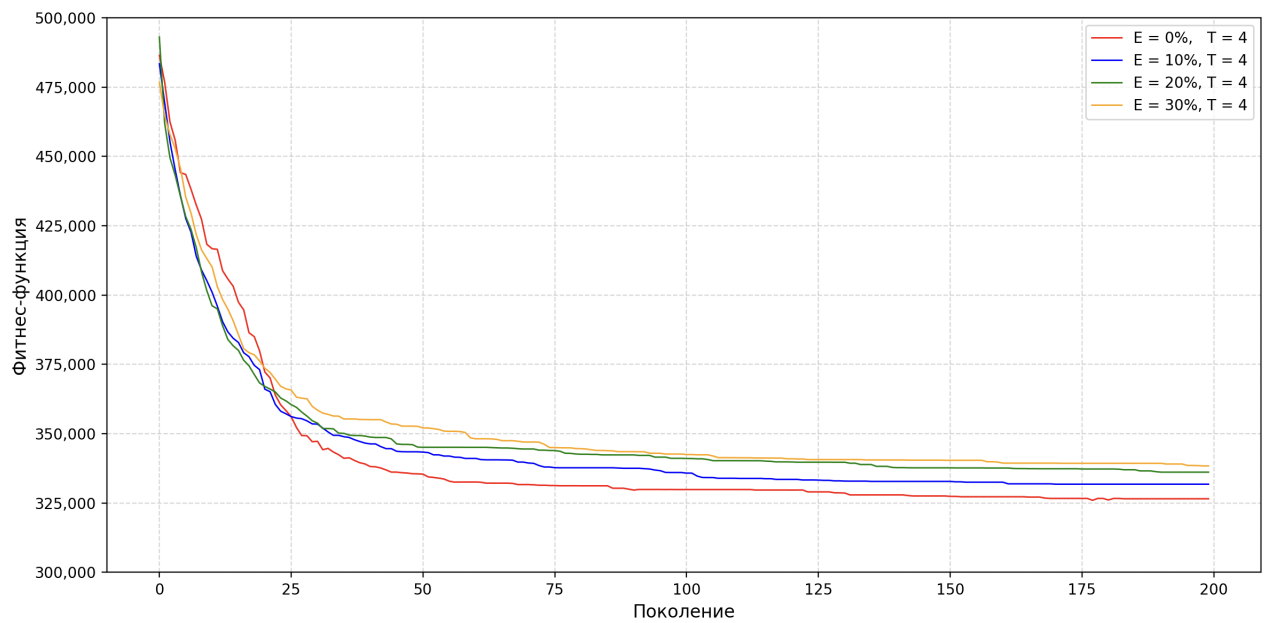


Рисунок 26 – График сходимости зависимости целевой функции от номера поколения при  $T = 4$

#### 4.4 Вывод

В результате проведенных исследований было доказано, что построенная модель является адекватной и позволяет получить близкое к оптимальному решение, а при малом количестве контейнеров генетический метод может получить такое же оптимальное решение, как и метод полного перебора. Также удалось установить, что такие параметры генетического метода, как размер турнира и процент элитных особей, находятся в тесной взаимосвязи и могут компенсировать друг друга. Исследование позволило найти комбинации данных параметров, при которых метод дает наиболее оптимальное решение.

## ЗАКЛЮЧЕНИЕ

В рамках данной работы был разработан метод для решения задачи упаковки контейнеров в трюм корабля и разработано программное обеспечение, реализующее данный метод.

Достигнута поставленная цель. Были выполнены следующие задачи:

- выбраны критерии и ограничения оптимизации упаковки контейнеров в трюм корабля, формализована постановка задачи в виде математической модели;
- проведен обзор существующих эвристических методов для решения оптимизационных задач, приведены результаты сравнительного анализа;
- модифицирован генетический алгоритм для решения поставленной задачи, представлены ключевые этапы метода в виде схем алгоритмов;
- создана программная реализация разработанного метода;
- проверена адекватность построенной модели, исследована эффективность реализованного метода при различных значениях параметров.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Shipping Container Types: A Guide. [Электронный ресурс]. — Режим доступа — URL: <https://dcsa.org/newsroom/shipping-container-types-a-guide> — (дата обращения: 25.11.2024).
2. Tupper E. C. Introduction to Naval Architecture. — Elsevier Ltd., 2013. Р. 33–46.
3. Реконфигурация географии торговли и цепочек поставок: последствия для торговли, глобальных цепочек создания стоимости и морского транспорта. [Электронный ресурс]. — Режим доступа — URL: [https://unctad.org/system/files/official-document/cid54\\_ru.pdf](https://unctad.org/system/files/official-document/cid54_ru.pdf) — (дата обращения: 18.12.2024).
4. Container ships. [Электронный ресурс]. — Режим доступа — URL: <https://www.ics-shipping.org/explaining/ships-ops/container-ships/> — (дата обращения: 26.11.2024).
5. Lee C., Song D. Ocean container transport in global supply chains: Overview and research opportunities. — Transportation Research Part B — 2017 — 453–456 с.
6. Тимофеева О.П., Чернышева Т.Ю., Корелин О.Н., Волков А.В. Генетический алгоритм в оптимизации трехмерной упаковки блоков в контейнер // Труды НГТУ им. Р. Е. Алексеева, 2017. №2 (117). URL: <https://cyberleninka.ru/article/n/geneticheskiy-algoritm-v-optimizatsii-trehmernoyp-upakovki-blokov-v-konteyner> (дата обращения: 12.12.2024).
7. Курейчик В. В., Глущенко А. Е., Орлов А. Н. Гибридный подход для решения задачи 3-х мерной упаковки // Известия ЮФУ. Технические науки, 2016. №6 (179). URL: <https://cyberleninka.ru/article/n/gibridnyy-podhod-dlya-resheniya-zadachi-3-h-mernoyp-upakovki> (дата обращения: 15.12.2024).
8. Wang, Yizhun. Review on greedy algorithm // Theoretical and Natural Science, 22023, 14, 233-239. URL:

- [https://www.researchgate.net/publication/376076453\\_Review\\_on\\_greedy\\_algorithm](https://www.researchgate.net/publication/376076453_Review_on_greedy_algorithm) (дата обращения: 27.12.2024).
9. Nikolaev A.G., Jacobson S. Simulated Annealing // Handbook of Metaheuristics, 2010, 0.1007/978-1-4419-1665-5\_1. URL: [https://www.researchgate.net/publication/226778564\\_Simulated\\_Annealing](https://www.researchgate.net/publication/226778564_Simulated_Annealing) (дата обращения: 15.12.2024).
  10. Kumar M. Genetic Algorithm - an Approach to Solve Global Optimization Problems // Indian Journal of Computer Science and Engineering. 2010. №1. 199-206. URL: [https://www.researchgate.net/publication/283361244\\_Genetic\\_Algorithm\\_-\\_an\\_Approach\\_to\\_Solve\\_Global\\_Optimization\\_Problems](https://www.researchgate.net/publication/283361244_Genetic_Algorithm_-_an_Approach_to_Solve_Global_Optimization_Problems) (дата обращения: 17.12.2024).
  11. Dorigo M., Birattari M., Stützle T. Ant Colony Optimization // Computational Intelligence Magazine, 2006, IEEE. 1. 28-39. 10.1109/MCI.2006.329691. URL: [https://www.researchgate.net/publication/308953674\\_Ant\\_Colony\\_Optimization](https://www.researchgate.net/publication/308953674_Ant_Colony_Optimization) (дата обращения: 12.12.2024).
  12. Dornas A., Sarubbi J., Martins F., Wanner E. Real-Polarized Genetic Algorithm for the Three-Dimensional Bin Packing Problem // Annual Conference on Genetic and Evolutionary Computation. 2017. URL: [https://publications.aston.ac.uk/id/eprint/31436/1/Real\\_polarized\\_genetic\\_algorithm\\_for\\_the\\_three\\_dimensional\\_bin\\_packing\\_problem.pdf](https://publications.aston.ac.uk/id/eprint/31436/1/Real_polarized_genetic_algorithm_for_the_three_dimensional_bin_packing_problem.pdf) (дата обращения: 01.04.2025).
  13. Goncalves J., Resende M. A Biased Random-Key Genetic Algorithm for a 2D And 3D Bin Packing Problem // International Journal of Production Economics. 2013. URL: <https://mauricio.resende.info/doc/brkgabinpacking.pdf> (дата обращения: 03.04.2025).
  14. Документация по языку C. [Электронный ресурс]. — Режим доступа — URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/> — (дата обращения: 10.04.2025).
  15. Avalonia documentation. [Электронный ресурс]. — Режим доступа — URL: <https://docs.avaloniaui.net> — (дата обращения: 18.04.2025).

16. Размеры морских контейнеров. [Электронный ресурс]. — Режим доступа — URL: <https://foot-container.ru/razmery-morskih-kontejnerov/> — (дата обращения: 05.05.2025).
17. Non-Average Shipping Container Sizes. [Электронный ресурс]. — Режим доступа — URL: <https://www.tradecorp-usa.com/blog/non-average-shipping-container-sizes-16-24-30-45-48-etc/> — (дата обращения: 06.05.2025).
18. Контейнеровозы. [Электронный ресурс]. — Режим доступа — URL: <https://shipsforsale.su/catalog/container/> — (дата обращения: 06.05.2025).

## **ПРИЛОЖЕНИЕ А**

Презентация к научно-исследовательской работе состоит из 13 слайдов.