МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки / специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль) /специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Программная реализация решения логистических задач на основе генетического алгоритма

Обучающийся М.В. Демченко
(Инициалы Фамилия) (личная подпись)

Руководитель к. т. н, доцент, Н.А. Сосина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант к. ф. н, доцент, М.В. Дайнеко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Тема данной выпускной квалификационной работы: «Программная реализация решения логистических задач на основе генетического алгоритма».

ВКР состоит из введения, четырёх глав, 11 рисунков, 5 таблиц, заключения и списка из 23 источников, включая иностранные.

Целью ВКР является разработка и изучение гибкого метода решения несбалансированных транспортных задач произвольной размерности, способного эффективно справляться с ограничениями логистических систем.

Объектом ВКР является транспортная оптимизация в логистике.

Предметом ВКР является реализация генетического алгоритма с островной моделью.

Ключевой задачей является создание масштабируемого инструмента для решения сложных транспортных задач.

Работа включает несколько логически связанных частей: теоретический обзор транспортных задач, методов оптимизации и эволюционных алгоритмов, архитектуру программной системы и её тестирование.

Первая глава содержит обзор транспортных задач и сравнительный анализ классических и метаэвристических методов. Во второй главе описаны этапы генетического алгоритма и его применение к задачам высокой размерности. Третья глава посвящена архитектуре системы, особенностям реализации и интерфейсу. В четвёртой главе приведены результаты тестирования, сравнение с классическими методами и решателем HiGHS, а также визуализация решений.

В заключение подчеркивается, что разработанная программа обеспечивает гибкое и эффективное решение сложных логистических задач.

Работа представляет интерес для широкого круга читателей, занимающихся логистикой, исследованием операций и прикладной оптимизацией.

Abstract

The present graduation work is devoted to software implementation of solving logistics problems using a genetic algorithm.

The graduation work consists of an introduction, 4 parts, 11 figures, 5 tables, a conclusion, and a list of 23 references including foreign sources.

The aim of this research is to develop and explore a flexible method of solving unbalanced transportation problems of arbitrary dimension, capable to effectively cope with the limitations of the logistics systems.

The object of the study is the process of transport optimization in the logistics systems.

The subject of the graduation work is the design and implementation of the adaptive genetic algorithm for solving multidimensional transport problems.

The key issue of the graduation work is the creation of a scalable and universal tool for solving the logistics problems with arbitrary structure.

The graduation work may be divided into several logically connected parts which are: a theoretical review of the transportation problems and the optimization methods, discussion of the evolutionary algorithms and the software architecture as well as its testing.

The first part provides an overview of the transportation problems and a comparative analysis of the classical and metaheuristic methods. The second part dwells on the stages of the genetic algorithm and discusses its application to high-dimensional problems. The third part describes the system architecture, the implementation details, and the interface. The fourth part presents the testing results, performance on various tasks, comparison with the classical methods and HiGHS solver, as well as the visualization of the results.

In conclusion, it is emphasized that the developed software ensures flexible and effective solutions to the complex logistics problems.

The work is of interest for a wide circle of readers engaged in logistics, operation research, and applied optimization.

Оглавление

Введение

Оптимизация распределения ресурсов является одной из центральных задач в прикладной математике и экономике. Одним из наиболее известных и широко применяемых её частных случаев является транспортная задача, описывающая процесс минимизации затрат на доставку продукции от поставщиков к потребителям при заданных объемах поставок и спроса. Классическая постановка этой задачи предполагает баланс между общими объёмами предложения и спроса. Однако в реальных прикладных задачах, особенно в сфере логистики, торговли и управления цепочками поставок, это условие выполняется крайне редко.

На практике чаще возникают несбалансированные транспортные задачи, в которых суммарный объём поставок не равен суммарному спросу. Это может быть связано с издержками производства, стратегическими резервами, колебаниями спроса или форс-мажорными ситуациями. Решение таких задач требует адаптации стандартных алгоритмов или использования универсальных подходов, способных эффективно справляться с несимметричными условиями.

Кроме того, при усложнении логистических процессов может возникать необходимость учитывать дополнительные параметры, такие как промежуточные склады, временные ограничения, разновидности грузов и другие факторы. Такие задачи приводят к многоиндексным (многомерным) обобщениям классической транспортной задачи, где количество индексов превышает два, а размерность пространства решений значительно возрастает. Это делает традиционные методы вычислительно затратными и ограниченно применимыми в крупных бизнес-сценариях.

В связи с этим особую актуальность приобретает применение эволюционных подходов, в частности, генетических алгоритмов, способных работать с плохо формализованными задачами, множественными ограничениями и сложной структурой пространства решений. Эти алгоритмы

могут быть адаптированы к различным условиям, включая несбалансированные случаи и задачи произвольной размерности.

В настоящей выпускной квалификационной работе рассматриваются методы решения как классической, так и многоиндексной транспортной задачи с акцентом на несбалансированные постановки, наиболее характерные для реальных экономических процессов. Предусматривается разработка универсального программного решения с графическим интерфейсом пользователя, средствами визуализации и возможностью экспорта результатов. Основу метода составит генетический алгоритм, дополненный механизмами генерации начальной популяции с использованием классических методов (в том числе метода северо-западного угла, минимального элемента, случайного заполнения) и промышленного решателя линейных задач HiGHS.

Цель работы заключается в разработке и исследовании гибкого метода решения несбалансированных транспортных задач произвольной размерности, способного эффективно справляться с ограничениями и сложностями, возникающими при моделировании реальных логистических процессов.

Для достижения поставленной цели решаются следующие задачи.

Анализ существующих подходов к решению многоиндексной транспортной задачи и методов оптимизации.

Разработка универсального представления входных данных, пригодного для задач произвольной размерности.

Реализация генетического алгоритма с островной моделью и многоуровневой инициализацией популяции.

Разработка графического интерфейса для работы с задачами, отображения решений и экспорта результатов.

Проведение сравнительного анализа применимого для создания в ВКР программного продукта метода с классическими алгоритмами (метод северо-западного угла, метод минимального элемента) и решателем HiGHS.

1 Теоретические и прикладные основы оптимизации логистических и транспортных задач

1.2 Логистические задачи и их особенности

Логистика представляет собой междисциплинарную область, ориентированную на управление материальными, информационными и финансовыми потоками с целью обеспечения эффективного распределения ресурсов. Основная задача логистики заключается в оптимизации всех этапов движения товаров и услуг – от производства до их конечного потребителя. Решение возникающих в данной сфере задач требует применения методов математического моделирования и анализа, что обусловлено сложностью и многогранностью логистических процессов.

С точки зрения математического моделирования, логистические задачи можно классифицировать на несколько основных категорий:

– задачи управления запасами, направленные на оптимизацию объёмов хранения с учётом динамики спроса, логистических издержек, сроков поставок и т.д.;

– задачи складской логистики, включающие распределение товаров внутри складских комплексов, а также минимизацию временных и финансовых затрат на обработку грузов;

– задачи транспортной логистики, связанные с планированием и организацией перевозок грузов или пассажиров между пунктами назначения.

Несмотря на разнообразие логистических задач, многие из них сводятся к единой ключевой проблеме – оптимизации транспортных процессов, поскольку транспортная составляющая является определяющим фактором эффективности логистических операций. Доля транспортных расходов в общей структуре логистических затрат может быть значительной, что обуславливает необходимость разработки и внедрения оптимизационных

методов для снижения издержек при соблюдении существующих ограничений (вместимость транспортных средств, временные ограничения, маршрутизация).

Транспортные задачи, являясь частными случаями логистических задач, приобрели статус самостоятельного объекта исследования благодаря их практической значимости и строгой математической формализации. Последняя позволяет эффективно применять методы математического программирования и алгоритмы оптимизации, что делает транспортные задачи удобными для анализа и моделирования. Разработанные для их решения подходы отличаются высокой адаптивностью и находят применение не только в сфере транспортной логистики, но и в смежных областях, включая управление цепями поставок и распределение ресурсов. Использование современных методов, таких как генетические алгоритмы и другие метаэвристические подходы, открывает возможности для получения более качественных решений по сравнению с традиционными методами оптимизации, что особенно актуально в условиях усложняющихся логистических процессов.

С учётом вышеизложенного, в данной работе рассматривается транспортная задача маршрутизации транспортных средств, обладающая высокой прикладной значимостью. Исследование данной задачи позволит продемонстрировать эффективность применения генетического алгоритма для её решения, а также выявить его преимущества и недостатки по сравнению с классическими методами оптимизации и промышленным решателем HiGHS.

1.2 Классификация и методы решения транспортных задач

Транспортные задачи представляют собой один из фундаментальных классов задач дискретной оптимизации, направленных на определение наилучшего способа распределения потоков между поставщиками и потребителями при минимальных совокупных затратах. Являясь частным

случаем задач линейного программирования, они обладают особой структурой, позволяющей применять специализированные методы, обеспечивающие высокую вычислительную эффективность. Первоначально возникнув в контексте планирования перевозок, транспортные задачи в настоящее время находят широкое применение в логистике, производственных системах, телекоммуникациях, распределённых вычислениях и других областях.

Классическая формулировка транспортной задачи заключается в минимизации затрат на перевозку однородного ресурса от множества поставщиков к множеству потребителей. При этом известны объёмы предложения и спроса, а также стоимость перевозки единицы ресурса между каждым источником и пунктом назначения. При выполнении условия сбалансированности (сумма предложений равна сумме потребностей) задача допускает допустимое решение и может быть эффективно решена с использованием линейного программирования, в том числе с применением специализированных алгоритмов – например, метода северо-западного угла, метода минимального элемента и метода потенциалов.

Однако в реальных прикладных задачах чаще встречаются несбалансированные постановки, в которых общий объём предложения не равен совокупному спросу. Подобные случаи типичны для логистических систем, где часть ресурсов может остаться невостребованной или, наоборот, потребности превышают доступные объёмы. В таких ситуациях задача модифицируется с добавлением фиктивного поставщика или потребителя, что позволяет сохранить применимость классических методов, однако усложняет интерпретацию полученных решений и может снижать их устойчивость к изменениям входных данных.

Значительно более сложную структуру имеют многоиндексные транспортные задачи, в которых перемещение ресурсов зависит не только от источников и пунктов назначения, но и от дополнительных факторов – времени, типа продукции, вида транспорта и других параметров. Такие задачи

формализуются в виде многомерных массивов и требуют оптимизации в пространствах высокой размерности. Применение классических методов в этих условиях оказывается невозможным, что обуславливает необходимость использования универсальных или эвристических алгоритмов, способных справляться с многомерной структурой и сложной системой ограничений.

Существуют и другие обобщённые постановки транспортных задач. В частности, в стохастических транспортных моделях параметры задачи (например, объёмы предложения, спрос или издержки) описываются случайными величинами, отражающими неопределённость реальных логистических процессов. Решение таких задач осуществляется методами стохастического программирования, сценарного анализа и робастной оптимизации. Кроме того, в динамических транспортных задачах учитывается изменение параметров во времени, что требует оптимизации с учётом временной последовательности действий.

С теоретической точки зрения транспортные задачи относятся к задачам на потоках в графах, что позволяет использовать методы сетевого программирования – алгоритмы минимального пути, максимального потока, минимального разреза и другие. Однако при добавлении нелинейных издержек, временных окон, приоритезации маршрутов и других реалистичных ограничений задача становится NP-трудной, а точное решение в общем случае невозможно за полиномиальное время.

В современной научной и инженерной практике особое внимание уделяется применению эвристических и метаэвристических методов, способных эффективно решать крупномасштабные и слабо структурированные транспортные задачи. Среди них наибольшую популярность получили методы эволюционных вычислений, включая генетические алгоритмы, которые позволяют получать приближённые решения высокого качества при ограниченных вычислительных ресурсах. Это делает их особенно востребованными в задачах с высокой размерностью, множеством ограничений и нестабильными входными данными.

Таким образом, транспортные задачи, несмотря на внешнюю простоту формулировки, охватывают широкий спектр теоретических и прикладных направлений. Их эффективное решение требует комплексного подхода, сочетающего методы линейного программирования, теории графов, вероятностного анализа и вычислительного интеллекта.

1.3 Сравнение методов решения транспортной задачи

Оптимизация транспортных процессов представляет собой фундаментальное направление в прикладной математике и операционном исследовании. Выбор метода решения транспортной задачи определяется не только её размерностью, но и сложностью ограничений, степенью структурированности данных, требованиями к точности, а также допустимым временем вычислений. Современная практика показывает, что для достижения наилучших результатов рационально сочетать традиционные подходы и современные эвристические методы, адаптируя выбор алгоритма к конкретным условиям задачи.

Классические методы, применяемые при решении транспортных задач, основываются на строгих математических процедурах, позволяющих находить точные решения в детерминированной постановке. Метод северо-западного угла, метод минимальной стоимости и метод потенциалов являются основными представителями данной группы. Они демонстрируют высокую эффективность при малых и средних размерах задачи, а также при строгой сбалансированности входных данных и наличии линейной структуры ограничений. Тем не менее, эти методы существенно теряют производительность и вычислительную устойчивость при усложнении модели, в том числе при появлении дополнительных ограничений, несбалансированности и недопустимости базисных решений.

Одним из эффективных инструментов точной оптимизации транспортных задач является использование промышленных решателей

линейного программирования, таких как HiGHS. В отличие от ручных классических алгоритмов, данный метод реализует современные версии симплекс-метода, внутренней точки и ветвей и границ (branch-and-bound), демонстрируя высокую точность, масштабируемость и производительность. Метод HiGHS особенно актуален при решении несбалансированных задач, в которых объёмы поставок и потребностей не совпадают, что типично для реальных логистических сценариев, и требует строгого учёта штрафных расходов или фиктивных узлов.

Наряду с точными методами, значительную роль в решении транспортных задач играют эвристические и метаэвристические алгоритмы, применяемые для получения приближённых решений в условиях высокой размерности или частичной неопределённости исходных данных. Генетический алгоритм, муравьиный алгоритм, алгоритмы роя частиц и другие стохастические подходы активно развиваются в последние десятилетия и позволяют эффективно находить решения в условиях, где традиционные методы становятся непрактичными из-за чрезмерной вычислительной сложности. Основное преимущество метаэвристик заключается в способности преодолевать локальные минимумы и адаптироваться к структуре задачи, хотя при этом они не гарантируют нахождение глобального оптимума.

Сравнительная характеристика методов представлена в таблице 1.

Таблица 1 – Сравнение методов решения транспортных задач

Метод	**Преимущества**	**Недостатки**	**Сложность**	**Область применения**
Метод северо-западного ула	Простота реализации, быстрое получение допустимого базисного решения	Не всегда даёт оптимум, требует последующей оптимизации	O(m+n)	Начальный этап, базисное решение для малых задач

Продолжение таблицы 1

Метод	**Преимущества**	**Недостатки**	**Сложность**	**Область применения**
Метод минимальной стоимости	Простота реализации, более выгодное базисное решение по сравнению с СЗУ	Не гарантирует оптимальность, требуется дооптимизация	O(m·n)	Малые и средние задачи, построение начального решения
Метод потенциалов	Формализованность, точное решение	Вычислительно затратен при больших размерностях	$O(m \cdot n^2)$	Малые и средние задачи
Распределительный метод	Не требует начального плана, может дать глобальный оптимум	Сложен в реализации, чувствителен к данным	$O(m \cdot n^2)$	Уточнение решений, задачи средней сложности
Муравьиный алгоритм	Избегает локальных минимумов, подходит для маршрутизации	Высокая вычислительная сложность, требует настройки параметров	Высокая	Сложные, динамические и маршрутизационные задачи
Генетический алгоритм	Универсальность, работа с ограничениями, масштабируемость	Зависимость от параметров, отсутствие гарантии оптимальности	Высокая	Большие задачи, задачи с множеством ограничений
Решатель HiGHS	Высокая точность, промышленное качество, поддержка несбалансированных задач	Меньшая гибкость в нестандартных ограничениях, чувствительность к формализации	Зависит от метода (LP/ILP)	Промышленные задачи, большие и несбалансированные модели

Таким образом, теоретический анализ подтвердил актуальность задач оптимизации транспортных процессов в логистике, выявил ограничения классических методов при решении сложных практических задач и обосновал выбор генетического алгоритма в качестве перспективного метода для их решения, что определяет направление дальнейшего исследования в данной работе.

2 Теоретические основы алгоритмов оптимизации

2.1 Эволюционные алгоритмы и их применение в логистике

Эволюционные алгоритмы (ЭА) представляют собой класс стохастических методов оптимизации, основанных на принципах естественного отбора, генетической рекомбинации и мутаций, что позволяет эффективно решать задачи с высокой размерностью и сложными ограничениями. Данный подход является универсальным инструментом для поиска приближённых оптимальных решений в широком спектре прикладных областей, в том числе и в логистике.

Основная идея эволюционных алгоритмов заключается в моделировании биологических процессов эволюции, где популяция кандидатов-решений (индивидов) постепенно улучшается посредством последовательных поколений. Каждый индивид представляется в виде хромосомы, структура которой кодирует потенциальное решение задачи.

Ключевыми составляющими эволюционных алгоритмов являются селекция, кроссовер и мутация. Процесс селекции обеспечивает выбор решений с наивысшей приспособленностью, что способствует усилению влияния качественных вариантов и направленному улучшению популяции. Рекомбинация, осуществляемая посредством кроссовера, позволяет объединить генетический материал двух или более родительских решений, что порождает новые, потенциально более оптимальные комбинации признаков. Введение случайных изменений, то есть мутация, сохраняет генетическое разнообразие и помогает алгоритму избегать застревания в локальных оптимумах.

Логистические задачи, характеризующиеся высокой сложностью, множественностью ограничений и изменчивостью исходных данных, требуют гибких и адаптивных методов решения. Эволюционные алгоритмы демонстрируют преимущества в данном контексте благодаря устойчивости к

локальным минимумам. Кроме того, современные вычислительные платформы позволяют реализовать данные методы в параллельном режиме, что особенно актуально при решении задач большого объёма.

Практическое применение эволюционных алгоритмов в логистике охватывает ряд направлений, таких как оптимизация маршрутов доставки, планирование перевозок, распределение ресурсов между узлами транспортной сети и управление запасами. Например, при решении задачи маршрутизации транспортных средств эволюционные алгоритмы позволяют найти решения, минимизирующие суммарные затраты на перевозку, одновременно удовлетворяя множеству логистических ограничений. При этом гибкость алгоритма обеспечивает адаптацию к динамическим изменениям в условиях транспортировки и спроса.

Ряд исследований (Голдберг, 1989; Michalewicz, 1996) демонстрирует, что эволюционные алгоритмы, включая генетические алгоритмы, успешно применяются для решения задач оптимизации, связанных с логистическими процессами. В частности, их способность комбинировать жадные эвристики с глобальным поиском оптимального решения позволяет достигать высоких показателей эффективности при сравнении с традиционными методами, такими как метод северо-западного угла или метод потенциалов.

Также, применение эволюционных алгоритмов в логистике находит своё отражение в деятельности ряда российских компаний, стремящихся повысить эффективность управления маршрутами, распределением ресурсов и оптимизацией затрат. Благодаря своей гибкости и способности учитывать сложные ограничения, эволюционные алгоритмы позволяют адаптировать решения к специфике каждого логистического процесса.

На российском рынке наблюдается тенденция к внедрению облачных оптимизирующих платформ, реализующих данный подход. Примером является компания Veeroute, которая с 2014 года занимается разработкой облачного движка комбинаторной оптимизации. Платформа Veeroute интегрирует математическое ядро, специализированные API и модули

маршрутизации для решения широкого спектра логистических задач, таких как доставка последней мили, распределение заказов и планирование маршрутов. Благодаря использованию алгоритмов, учитывающих такие параметры, как дорожная ситуация, плотность трафика и региональные особенности, клиенты компании достигают снижения затрат на топливо, сокращения времени доставки и повышения качества обслуживания.

Кроме того, наблюдается активное развитие нишевых решений в сфере логистики, ориентированных на оптимизацию работы складских комплексов, диспетчеризацию и автоматизацию процессов последней мили. В данном контексте малые и средние IT-компании разрабатывают специализированные TMS-системы (Transportation Management System), в основе которых лежат эволюционные алгоритмы, позволяющие в режиме реального времени адаптироваться к изменяющимся условиям транспортной среды. Такие системы демонстрируют высокую адаптивность: операторы мутаций и скрещивания генерируют новые комбинации решений, что позволяет алгоритмам успешно выходить из локальных оптимумов и находить более эффективные маршруты при изменении параметров городской инфраструктуры.

Применение эволюционных алгоритмов в логистике характеризуется рядом существенных преимуществ. Во-первых, алгоритмическая адаптивность обеспечивает устойчивость к внешним изменениям, что особенно важно в условиях динамичного городского движения и сезонных колебаний спроса. Во-вторых, возможность быстрого получения приближённых оптимальных решений позволяет значительно сократить вычислительные затраты и время, необходимое для планирования маршрутов, что приводит к снижению общих операционных расходов. В-третьих, облачные решения с открытыми API способствуют интеграции оптимизирующих платформ с существующими CRM- и TMS-системами, обеспечивая комплексное управление логистической цепочкой.

Таким образом, опыт российских компаний, разрабатывающих

облачные оптимизирующие платформы, свидетельствует о высокой практической значимости эволюционных алгоритмов в логистике. Применение данных методов позволяет оптимизировать сложные транспортно-логистические процессы, учитывая многочисленные ограничения и специфику регионального рынка, что приводит к снижению затрат, повышению эффективности распределения ресурсов и оперативности принятия решений. В дальнейшем развитие и адаптация эволюционных алгоритмов будут способствовать укреплению конкурентоспособности предприятий малого и среднего бизнеса, делая их незаменимым элементом современной системы логистического управления.

2.2 Преимущества и ограничения генетического алгоритма

Рассмотрев особенности применения генетического алгоритма в транспортных задачах, целесообразно остановиться на его ключевых характеристиках, оказывающих влияние на качество и устойчивость получаемых решений. Данный метод, широко используемый в задачах комбинаторной оптимизации, проявляет ряд свойств, обеспечивающих его актуальность в контексте моделирования логистических систем.

Прежде всего, генетический алгоритм выгодно отличается способностью работать в условиях неполной или не формализуемой информации о структуре задачи. В отличие от строгих математических методов, он не требует линейности функции цели или ограничений, что позволяет эффективно применять его в задачах с дополнительными параметрами, нестандартными зависимостями или многоиндексной структурой. Такая гибкость делает возможной его интеграцию в модель, описанную в предыдущем разделе, без существенной переработки алгоритма.

Важно отметить и его способность к исследованию пространства решений за счёт стохастических операторов – мутации и кроссовера. Это позволяет избегать преждевременной сходимости и повышает вероятность

нахождения приближённого глобального решения, особенно в случаях, когда пространство допустимых вариантов содержит множество локальных экстремумов. В условиях реальных транспортных задач, таких как планирование поставок с учётом времени и стоимости, такое свойство становится критически важным.

Дополнительным фактором, подтверждающим применимость генетического алгоритма, является его высокая масштабируемость. Благодаря возможности параллельной генерации и обработки популяций решений, метод сохраняет эффективность даже при увеличении размерности входных данных. Это качество особенно востребовано в задачах, где классические методы демонстрируют значительное увеличение времени расчёта или утрату точности.

Вместе с тем, успешное применение генетического алгоритма требует внимательного подхода к его настройке. Значительное влияние на результат оказывает выбор параметров – размеров популяции, вероятностей операторов, глубины вычислений. При недостаточно сбалансированной конфигурации возможно либо преждевременное вырождение популяции, либо чрезмерные затраты времени на достижение приемлемого результата. Кроме того, эффективность метода напрямую зависит от корректного выбора формы представления (кодировки) решений: в транспортных задачах ошибочная кодировка может приводить к генерации некорректных или неинформативных маршрутов.

Таким образом, генетический алгоритм представляет собой адаптивный и мощный инструмент, особенно эффективный в тех случаях, когда задача выходит за пределы применимости классических методов. Однако его результативность напрямую зависит от корректной архитектуры реализации и тщательной настройки параметров. В следующей главе будет рассмотрено, как генетический алгоритм сопоставляется по качеству и эффективности с другими подходами, включая традиционные и гибридные методы.

2.3 Генетический алгоритм как метод решения

Генетический алгоритм представляет собой итеративную эвристику, имитирующую процессы естественного отбора и эволюции в популяциях. Его основная идея заключается в последовательной генерации, отборе и преобразовании множества решений, где каждое из них интерпретируется как особь (индивид) в популяции. При решении транспортных задач – как классических, так и многоиндексных – каждый этап алгоритма требует особого подхода, поскольку напрямую влияет как на корректность формируемых решений, так и на эффективность поиска.

На начальном этапе производится генерация стартовой популяции. На практике этот процесс может быть реализован либо полностью случайным образом, либо с опорой на эвристические или точные методы. Случайное формирование обеспечивает высокое разнообразие, но в случае многоиндексной транспортной задачи приводит к существенным трудностям: из-за высокой размерности и усложнённых ограничений значительная часть решений оказывается не валидной и требует дополнительной обработки. Альтернативой служит гибридный подход, при котором часть популяции формируется с использованием решений, полученных методами северо-западного угла, минимального элемента, случайного допустимого построения или линейного решателя (например, HiGHS). Такая стратегия позволяет добиться сбалансированного начального распределения: с одной стороны, сохраняется разнообразие, а с другой – в популяции с самого начала присутствуют потенциально сильные решения. Сравнение описанных методов приведено в таблице 2. В контексте данной работы предпочтение отдано гибридной инициализации, что оправдано как с точки зрения вычислительной стабильности, так и с точки зрения ускорения сходимости.

Таблица 2 – Сравнение методов инициализации популяции

Метод инициализации	Преимущества	Недостатки
Полностью случайный	Высокое разнообразие решений	Много недопустимых особей, низкая сходимость
Эвристический (СЗУ, МинЭл)	Быстрая генерация допустимых решений	Малая вариативность
Решатель (HiGHS и др.)	Высокое качество начальных решений	Требует внешнего инструмента, низкое разнообразие
Гибридный подход	Комбинация точности и разнообразия	Увеличенная сложность реализации

После генерации особей необходимо оценить их приспособленность, т.е. соответствие целевой функции задачи. В классических транспортных задачах оценка может базироваться исключительно на общей сумме затрат, однако в многоиндексной постановке приходится учитывать целый ряд факторов: стоимость, временные ограничения, допустимость по всем индексам и часто – дополнительные условия, связанные с приоритетами, логистическими окнами или устойчивостью маршрутов. При этом важно не только вычислить значение целевой функции, но и определить меру штрафа за нарушения ограничений. Если такие нарушения игнорировать или недостаточно жёстко штрафовать, алгоритм может начать концентрироваться на формально дешёвых, но недопустимых решениях. Поэтому в разработанном подходе применяется модифицированная функция приспособленности, которая включает как нормированные значения целевых критериев, так и адаптивную систему штрафов. Это позволяет обеспечить универсальность при работе с задачами различной размерности и структуры, не перегружая при этом вычислительный процесс избыточными проверками на каждом поколении.

Модифицированная функция приспособленности может быть записана в виде:

$$F(x) = \mathrm{C}(x) + \sum_{i}^{N} \lambda_i * \phi_i(x), \quad (1)$$

где: *C(x)* – значение целевой функции для решения *x*,

N – количество ограничений,

$\phi_i(x)$ – функция штрафа за нарушение i-го ограничения,

λ_i – весовой коэффициент штрафа, адаптируемый в процессе выполнения алгоритма.

Следующим важным этапом является выбор операторов, определяющих поведение популяции в процессе эволюции. Оператор селекции отвечает за отбор особей, передающих свои характеристики следующему поколению. На практике применяются различные подходы: рулетка (пропорциональный отбор), турнирная селекция, отбор по рангу и другие. Каждый из них имеет как сильные, так и слабые стороны. Например, рулетка может страдать от преждевременной сходимости при сильной разнице в приспособленности, а турнирный отбор – усиливать селективное давление. При решении многоиндексной задачи особое значение приобретает баланс между сохранением разнообразия и усилением отбора, так как слишком агрессивная селекция может быстро уничтожить редкие, но потенциально ценные структуры, способные учитывать взаимодействия между несколькими индексами. В рамках практической реализации наиболее устойчивым оказался турнирный отбор с адаптивным размером турнира.

Операторы кроссовера (Рисунок 1) играют ключевую роль в передаче информации между родительскими решениями. Для задач с числовыми значениями применимы как стандартные одноточечные и многоточечные кроссоверы, так и специализированные операторы, учитывающие структуру решения, например, упорядочивающие или позиционные методы. Однако для многоиндексной задачи основной сложностью является сохранение допустимости потомков. Простая замена фрагментов между родителями

может привести к нарушению ограничений или появлению дублирующих распределений, что требует дополнительных механизмов коррекции. Это, в свою очередь, повышает вычислительную нагрузку, но без этих мер кроссовер теряет эффективность. Наиболее устойчивым оказался модифицированный одноточечный кроссовер с последующей корректировкой потомков по принципу минимального нарушения ограничений.

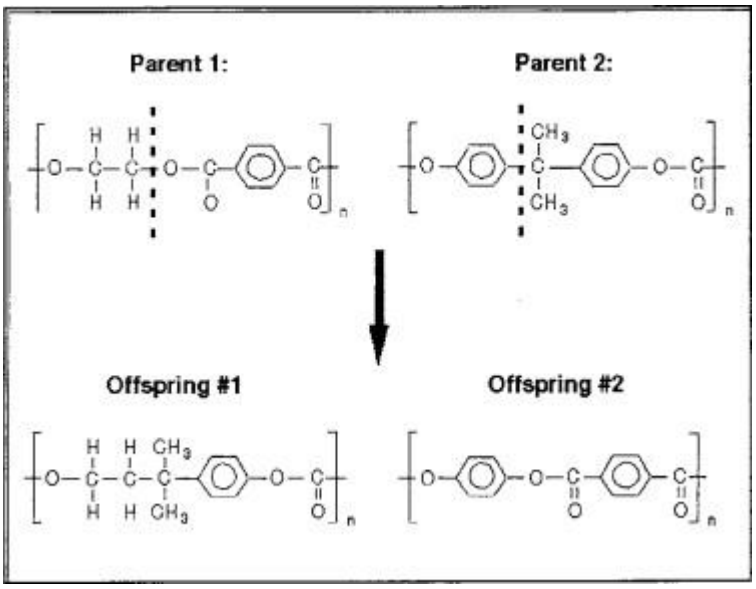


Рисунок 1 – Создание потомков из родительских решений (стандартный одноточечный кроссовер)

Оператор мутации используется для поддержания генетического разнообразия и исследования новых областей пространства решений. Наиболее распространены случайные изменения значений или перестановки элементов. Однако в контексте многоиндексной задачи важно, чтобы мутация не разрушала валидность решения. Это означает, что оператор должен быть достаточно «осторожным», модифицируя решение в пределах допустимых отклонений и обязательно соблюдая заданные балансы по всем индексам. Кроме того, мутация может использовать информацию о текущей структуре решения – например, встраивать элементы локального улучшения, повышая тем самым направленность эволюции. В ходе сравнений на практике наилучшие результаты показала направленная мутация с ограничением на

объём изменения и проверкой допустимости.

Контроль за процессом эволюции осуществляется с помощью стратегии замены поколений и условий остановки. Замена может быть полной (все родители заменяются потомками) или частичной (сохраняется часть лучших решений). В многоиндексной задаче сохранение элитных особей особенно важно, поскольку они могут представлять собой редкие, трудно достижимые комбинации индексов. Что касается условий остановки, традиционно используются фиксированное число поколений, отсутствие улучшений или достижение пороговой приспособленности. Однако на практике в многоиндексной задаче полезным оказывается комбинированный подход, позволяющий завершить работу алгоритма как при стагнации, так и по исчерпанию вычислительного бюджета.

В совокупности эти этапы формируют основу адаптивного генетического алгоритма, который способен справляться с высокой размерностью и структурной сложностью многоиндексных транспортных задач. Выбор конкретных операторов и параметров требует эмпирической настройки и многократного тестирования, поскольку универсальных решений не существует. Однако уже на уровне теоретического анализа видно, что корректная реализация каждого из этапов алгоритма имеет критическое значение для получения работоспособного и эффективного инструмента. Именно поэтому в следующей главе будет рассмотрена структура разрабатываемой системы и принятые проектные решения с учётом всех описанных особенностей.

3 Разработка алгоритмов для решения транспортных задач

Разработка программного решения для транспортной задачи, включая её многоиндексные модификации, потребовала создания гибкой и масштабируемой архитектуры, способной адаптироваться под различную размерность данных, методы решения и критерии оценки. Ключевой целью данной реализации стало объединение в одном программном продукте как классических алгоритмов, так и современных эволюционных подходов, с возможностью сравнения их качества и адаптации под практические сценарии. Особое внимание было уделено поддержке произвольной размерности задачи, а также обеспечению удобства пользовательского взаимодействия через графический интерфейс.

В ходе реализации была создана универсальная структура, позволяющая обрабатывать как классические двумерные, так и многоиндексные транспортные задачи. За счёт обобщения модели удалось свести обработку данных к единой форме, что обеспечило согласованную логику для всех методов решения, используемых в системе. Это позволило не только упростить поддержку кода, но и обеспечить единообразный вывод результатов, включая экспорт в таблицы, визуализацию планов перевозок и построение сравнительных графиков.

В данной главе будут подробно рассмотрены структура представления данных, особенности кодирования решений для генетического алгоритма, используемые методы генерации начальных решений, реализация операторов и механизмов эволюции, а также взаимодействие компонентов через интерфейс пользователя. Код самой программы приведен в приложении А и будет рассматриваться в этой главе лишь обобщенно.

3.1 Архитектура программного обеспечения и особенности его реализации

Представление исходных данных в программе строится на едином универсальном формате, способном описывать как классические, так и обобщённые транспортные задачи. Входные данные включают: размерность задачи, структуру индексов, массив стоимости перевозок, объёмы поставок и потребления, а также дополнительные параметры, определяющие ограничения или критерии оптимизации. Для реализации поддержки многоиндексных задач используется многомерная структура хранения, в виде тензора произвольной размерности, где каждая ось соответствует одному из измерений – например, отправителю, получателю, промежуточному складу, типу груза или временной метке.

В классическом случае, двумерная задача кодируется как матрица: каждая ячейка содержит стоимость перевозки от одного источника к одному потребителю. В обобщённой постановке, решение представляется в виде тензора, содержащего объёмы поставок по всем возможным комбинациям индексов. Например, в задаче размерности (3×3×3×3) тензор имеет четыре измерения, а в каждой точке – количество ресурса, отправляемого по соответствующему маршруту.

Такой подход потребовал разработки механизма валидации допустимости решений, проверяющего выполнение ограничений на каждом измерении. Для каждой оси задаются условия баланса (например, сумма по i-индексу должна соответствовать поставке с данного склада), и реализация проверяет их либо при генерации решения, либо после выполнения операторов кроссовера и мутации. Кроме того, используется система масок для исключения недопустимых маршрутов и ускорения проверок допустимости.

Хромосома в генетическом алгоритме реализована в виде одномерного массива, отображающегося на соответствующий тензор с помощью известной

размерности. Такой подход обеспечивает простоту реализации операторов ГА и поддержку произвольного количества измерений без необходимости переписывать отдельные алгоритмы для каждого случая.

В программе реализовано несколько методов генерации исходных допустимых решений, что позволяет формировать начальные планы перевозок для классического анализа, а также использовать их как элементы начальной популяции при запуске генетического алгоритма. Это существенно влияет на скорость сходимости и устойчивость вычислений, особенно в задачах с высокой размерностью.

Реализация метода северо-западного угла (СЗУ) базируется на пошаговом заполнении плана перевозок начиная с верхнего левого угла многомерной таблицы. Алгоритм последовательно распределяет объёмы поставок и потребления, продвигаясь по осевым координатам до исчерпания соответствующих ресурсов. В многоиндексной задаче реализация СЗУ усложняется: необходимо выбрать приоритетный порядок обхода индексов, чтобы обеспечить предсказуемое и корректное поведение на любом числе измерений. В программе это реализовано как итерация по координатам в лексикографическом порядке, что сохраняет алгоритмическую простоту и расширяет применимость.

Метод минимального элемента (LeastCost) построен на последовательном выборе ячеек с минимальной стоимостью среди ещё не распределённых, что обеспечивает, как правило, более выгодный по целевой функции стартовый план. Для многомерного случая был реализован механизм приоритетной сортировки всех координатных позиций по значению стоимости, с последующим последовательным распределением ресурсов. Этот метод, как и СЗУ, гарантирует допустимость решений и вносит начальное разнообразие в популяцию ГА.

Дополнительно реализован генератор случайных допустимых решений. В отличие от наивной случайной генерации, ведущей к множеству недопустимых решений, данный генератор использует ограниченное

перераспределение ресурса между допустимыми маршрутами с жёстким контролем баланса по всем измерениям. Это позволяет формировать допустимые, но разнообразные особи, пригодные для включения в популяцию без необходимости последующей фильтрации.

Также в состав системы включён интерфейс взаимодействия с внешним LP-решателем HiGHS, обеспечивающим построение эталонного плана перевозок при помощи линейного программирования. Это позволяет не только сравнивать точные и приближённые методы, но и использовать полученное решение как один из наиболее качественных источников для начальной популяции в ГА.

Благодаря сочетанию классических, эвристических и точных методов генерации решений обеспечивается гибкость начального этапа моделирования и закладывается основа для более устойчивой и быстрой работы эволюционного компонента.

Центральной частью программной системы является реализация генетического алгоритма, адаптированного для работы с транспортными задачами различной размерности. Алгоритм разработан таким образом, чтобы обеспечивать масштабируемость, устойчивость к локальным минимумам и возможность точной настройки параметров на разных классах задач.

Каждое решение (особь) в популяции кодируется в виде одномерного массива фиксированной длины, соответствующей количеству ячеек в многомерной структуре задачи. Это решение затем при необходимости преобразуется в тензор с заданной размерностью для вычисления целевой функции и проверки ограничений. Такой подход позволил унифицировать обработку хромосом и упростить реализацию операторов кроссовера и мутации, независимо от количества индексов. Все ограничения (например, по поставкам, потребностям, допустимости маршрутов и фиктивным направлениям) проверяются по маске допустимости и матрицам-ограничителям, генерируемым на этапе загрузки задачи.

Оценка приспособленности реализована как функция, включающая

значение целевой функции и штрафы за отклонения от допустимости. В процессе работы алгоритма штрафные коэффициенты могут изменяться: если множество допустимых решений слишком мало или популяция вырождается, штрафы ослабляются, чтобы расширить пространство поиска. Такая адаптивность повышает устойчивость алгоритма в сложных случаях.

Селекция реализована на основе турнирного отбора: случайно выбирается несколько особей, из которых в следующее поколение передаётся особь с наилучшей приспособленностью. Размер турнира, как и доля элитных особей, регулируется параметрами алгоритма. Это позволяет гибко управлять балансом между сохранением лучших решений и исследованием новых областей пространства.

Оператор скрещивания реализован в виде модифицированного одноточечного кроссовера. Для каждой пары родителей выбирается случайная точка разреза, после чего производится обмен сегментами. Поскольку такая операция может привести к нарушению ограничений, после кроссовера применяется механизм корректировки, выравнивающий поток по осям и исключающий дублирование. Дополнительно реализована проверка допустимости маршрутов, с автоматическим занулением запрещённых ячеек.

Мутация выполняется с заданной вероятностью и затрагивает случайные фрагменты особи. В большинстве случаев используется локальная направленная мутация – небольшое перераспределение объёмов между двумя допустимыми направлениями с сохранением общего баланса. При необходимости может применяться глобальная мутация – пересоздание целого блока решения, если популяция стагнирует. Этот механизм особенно важен для задач с большим количеством индексов, где даже незначительное нарушение структуры приводит к невыполнимым решениям.

Алгоритм использует частичную замену поколений: сохраняется заданное число лучших особей (элита), остальная часть заменяется новыми потомками. Такое решение позволяет избежать потери хороших решений и способствует стабилизации прогресса. Условия остановки включают

ограничение по числу поколений, достижение заданного уровня целевой функции и отсутствие улучшений за определённое число итераций.

Реализация поддерживает островную модель: популяция разбивается на изолированные подгруппы, каждая из которых эволюционирует независимо в течение нескольких поколений. Через определённые интервалы осуществляется обмен лучшими решениями между островами. Это повышает устойчивость к локальным минимумам и позволяет одновременно использовать различные стратегии в рамках одного запуска. На каждом острове может быть использован свой метод генерации стартовых решений, включая результаты классических алгоритмов и LP-решателя, что способствует сохранению разнообразия и ускоряет достижение оптимального результата.

Для повышения удобства работы с системой реализован графический интерфейс пользователя, позволяющий задавать параметры задачи, выбирать методы решения, отслеживать ход выполнения алгоритма и визуализировать полученные результаты. Интерфейс построен таким образом, чтобы быть интуитивно понятным и при этом обеспечивать доступ ко всем функциям, необходимым для настройки и анализа задач различной сложности.

На экране ввода пользователь указывает размерность задачи, вводит массивы поставок, потребностей и матрицу (или тензор) стоимостей перевозок. В случае многоиндексной задачи поля динамически адаптируются под число измерений. Дополнительно доступны настройки ограничений и фиктивных направлений, а также выбор целевой функции (например, по минимизации затрат или времени).

Для запуска расчёта пользователь может выбрать один из доступных методов: классический (северо-западный угол, минимальный элемент), линейный (HiGHS), случайный или генетический алгоритм. В случае выбора ГА становятся активны поля для задания параметров: размер популяции, вероятность мутации, число поколений, модель островов, механизм селекции и др.

В ходе работы алгоритма на экране отображается текущий план перевозок, значение целевой функции, лучшие найденные решения и график изменения целевой функции по поколениям. Также доступна функция досрочной остановки и фиксации текущего результата. Для задач с высокой размерностью доступна агрегированная визуализация, позволяющая быстро оценить структуру решения по проекциям.

После завершения расчёта программа предлагает экспортировать полученные данные в Excel-файл, включающий итоговый план перевозок, значение целевой функции и сравнительный отчёт по всем использованным методам. В файле формируется сводная таблица с минимальными затратами по каждому методу, а также графики сравнения эффективности на разных тестовых задачах. Для удобства анализа пользователь может выбрать формат представления (плоский, свёрнутый, агрегированный).

Благодаря такой организации интерфейса программа может использоваться не только в исследовательских целях, но и как инструмент поддержки принятия решений в логистике. Возможность обработки как классических, так и многоиндексных задач, интеграция с точным решателем и поддержка гибридных методов делают разработанную систему универсальным инструментом для анализа и оптимизации транспортных схем различной сложности.

3.2 Описание ключевых компонентов программного кода

Программная реализация построена по модульному принципу, что обеспечивает читаемость, масштабируемость и возможность повторного использования компонентов. Основной код реализован на языке Python и включает как графический интерфейс пользователя, так и вычислительное ядро, содержащее методы построения решений, проверки ограничений, выполнения генетического алгоритма и анализа результатов.

Структура проекта представлена в виде отдельных логически

выделенных модулей. Главный файл запускает интерфейс и связывает пользовательские действия с соответствующими функциями. Основные вычисления вынесены в отдельные модули: один отвечает за обработку входных данных и построение структуры задачи, другой – за реализацию алгоритмов, ещё один – за экспорт данных и построение графиков. Благодаря этому обеспечена изоляция логики визуализации от логики вычислений.

Модуль обработки данных включает функции создания многомерных тензоров, описывающих структуру задачи. Независимо от числа измерений, входные данные преобразуются в единый формат, поддерживающий проверку ограничений и генерацию допустимых маршрутов. Реализована система масок для исключения недопустимых направлений.

Методы классического решения, такие как метод северо-западного угла, минимального элемента, случайная генерация допустимого плана и подключение LP-решателя HiGHS, реализованы в виде отдельных функций с единым интерфейсом. Это позволяет запускать любой из них как отдельно, так и в составе генерации начальной популяции для генетического алгоритма. Вызов HiGHS осуществляется через формат LP-моделей, автоматически формируемый из текущей структуры задачи.

Центральное место в вычислениях занимает модуль генетического алгоритма. В нём реализованы функции генерации популяции, оценки приспособленности, селекции, кроссовера, мутации и отбора элитных особей. Хромосомы представлены в виде одномерных массивов с последующим преобразованием в многомерную структуру при проверке ограничений и расчёте стоимости. Используется адаптивная система штрафов, а также модифицированные операторы мутации и скрещивания, обеспечивающие допустимость потомков даже в многоиндексных задачах. Все параметры алгоритма (размер популяции, вероятности, стратегия элитизма, число поколений) доступны для изменения через интерфейс.

Реализована островная модель, позволяющая разбить популяцию на несколько независимых подгрупп. Каждая группа эволюционирует отдельно,

а через заданный интервал поколений происходит обмен лучшими решениями. Механизм реализован на уровне итеративной координации локальных популяций с контролем сходимости.

Модуль визуализации и экспорта формирует итоговые таблицы, строит графики изменения целевой функции и сравнения между методами, а также формирует Excel-файлы с результатами. Для графиков используется библиотека matplotlib, а для экспорта таблиц – openpyxl. Программа автоматически формирует итоговую сводку с минимальными значениями затрат по каждому методу и позволяет сравнивать их в табличной и графической форме.

Графический интерфейс реализован на базе tkinter. Окна автоматически адаптируются под размерность задачи: добавляются или скрываются соответствующие поля для многомерных индексов. Предусмотрены функции загрузки и сохранения данных, пошаговое выполнение алгоритма, отображение текущего транспортного плана, остановка расчёта и фиксация результата. Пользователь может задавать параметры алгоритма, переключать методы решения, получать мгновенную визуализацию прогресса и экспортировать отчёты.

Код программы структурирован таким образом, чтобы обеспечивать простоту поддержки, модификации и масштабирования. Все функции снабжены поясняющими комментариями. Архитектура ориентирована на расширение: возможно добавление новых методов, ограничений, стратегий отбора и визуализации без необходимости переписывать основную логику системы.

4 Тестирование и анализ результатов

В данной главе проводится экспериментальная проверка разработанного программного обеспечения, предназначенного для оптимизации транспортных маршрутов с использованием генетического алгоритма, а также сравнительный анализ его эффективности с классическими методами решения транспортных задач. Эксперименты выполнялись на различных тестовых наборах, отражающих разнообразные логистические сценарии.

4.1 Описание экспериментальной установки и тестовых данных

В данном параграфе изложена методика проведения экспериментов, направленных на оценку эффективности разработанного программного продукта, реализующего генетический алгоритм для решения транспортной задачи. Эксперименты проводились на вычислительном оборудовании, соответствующем типичным офисным стандартам, что позволяет оценить практическую применимость алгоритма в реальных условиях.

Тестирование осуществлялось в трех режимах: в первом режиме тестировался интерфейс, во втором режиме: для сбалансированной транспортной задачи, где затраты и возможности суммарно равны, и в третьем режиме для несбалансированной транспортной задачи, где затраты и возможности суммарно неравны. При этом параметры алгоритма, такие как размер популяции и количество поколений задаются через графический интерфейс, обеспечивая воспроизводимость экспериментов и возможность гибкой настройки метода в зависимости от условий конкретной задачи.

Исходными данными для экспериментов служат наборы, отражающие разнообразные логистические сценарии. В первую очередь, это данные о запасах поставщиков, спросе потребителей и матрице транспортных затрат. Для многоиндексной задачи дополнительно учитываются параметры, характеризующие вместимость складских помещений или иные ограничения,

влияющие на распределение ресурсов. Такой подход позволяет смоделировать как простые, так и сложные логистические процессы, соответствующие реалиям современного бизнеса.

Сравнение результатов работы разработанного алгоритма проводится по нескольким ключевым показателям, включая суммарные транспортные затраты, время вычислений и степень оптимизации распределения ресурсов. В качестве методов для сравнения выбраны классические методы: «минимальной стоимости» и «северо-западного угла» поскольку они широко применяется для получения базисного допустимого решения при решении транспортных задач. Планируется детальный анализ, который позволит оценить преимущества генетического алгоритма в условиях высокой размерности и сложных ограничений, а также выявить те сценарии, при которых классический метод может показаться более эффективным с точки зрения времени работы.

Ожидается, что предложенный алгоритм продемонстрирует более низкие суммарные затраты на перевозку и высокую адаптивность к изменяющимся условиям задачи, несмотря на увеличение времени вычислений. Проведение экспериментов позволит не только подтвердить практическую значимость разработанного программного продукта, но и выявить оптимальные настройки параметров, что, в свою очередь, станет основой для дальнейших усовершенствований алгоритма.

4.2 Интерфейс пользователя и функции программы

Разработанный программный продукт снабжён графическим интерфейсом, который позволяет запускать оптимизацию транспортной задачи без необходимости обращаться к коду. Интерфейс универсален: он одинаково поддерживает как классическую (двумерную), так и многоиндексную постановку задачи, обеспечивая единообразную работу независимо от размерности входных данных. Программа автоматически

определяет структуру задачи и применяет соответствующие алгоритмические процедуры, что избавляет пользователя от необходимости вручную переключать режимы.

Работа с системой начинается с загрузки входного файла формата JSON, содержащего матрицу затрат и массивы ограничений по каждой из осей – рисунок 2. После загрузки данные визуально отображаются в текстовом поле – отдельно выводятся ограничения и многомерная матрица затрат – рисунок 3. Это позволяет сразу убедиться в корректности структуры задачи и при необходимости внести изменения на этапе подготовки данных.

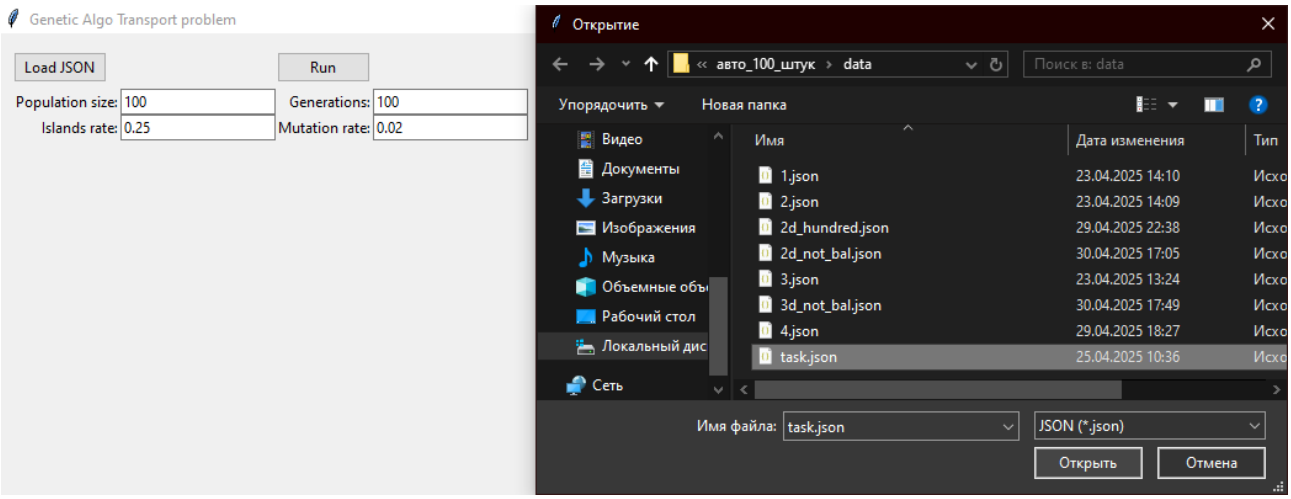


Рисунок 2 – Интерфейс программы, загрузка файла

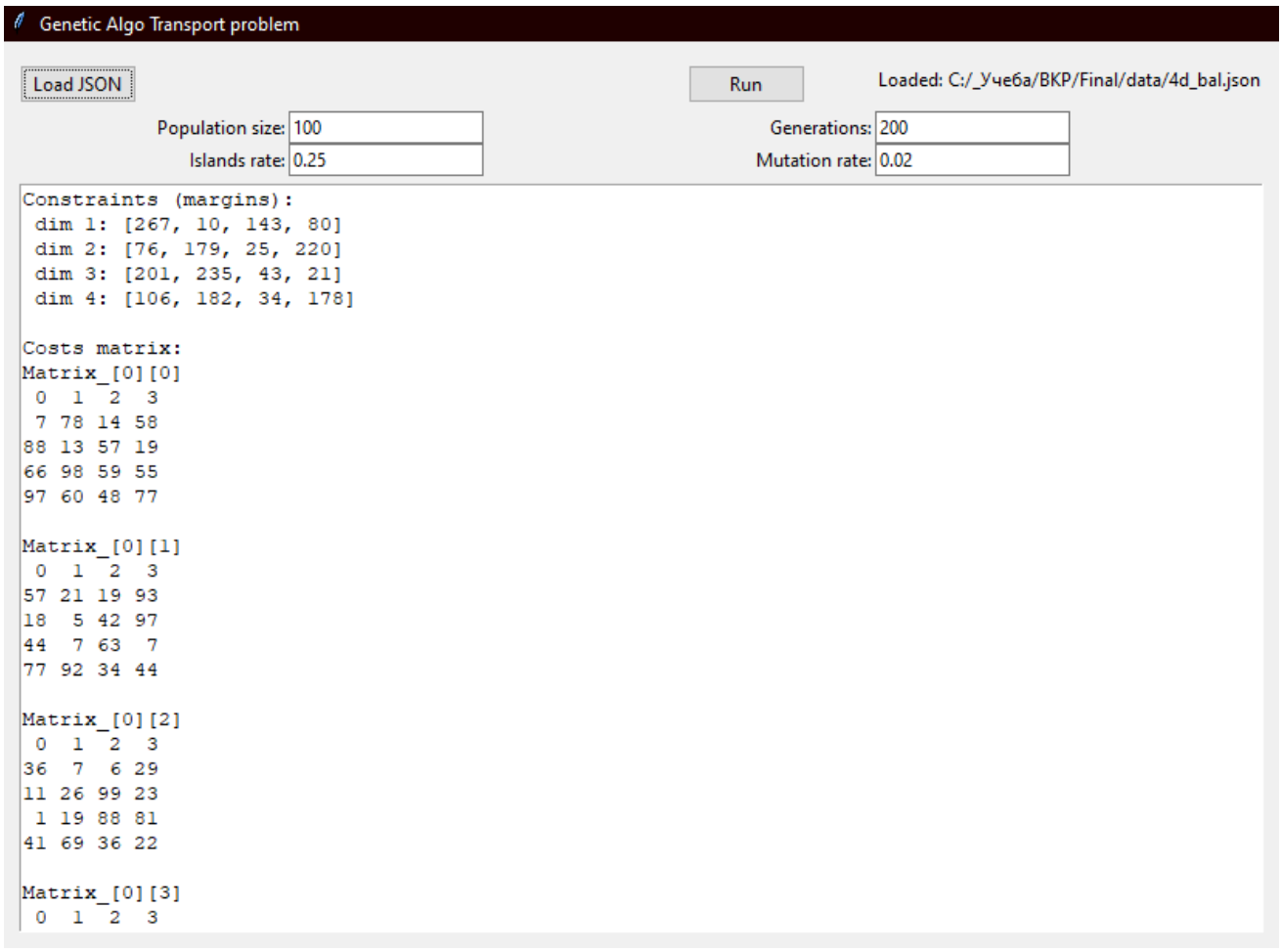


Рисунок 3 – Отображение загруженных данных

Пользователь задаёт параметры генетического алгоритма через соответствующие поля: размер популяции, количество поколений, вероятность мутации и долю особей, генерируемых с помощью классических методов (так называемых seed-методов) (рисунок 2, рисунок 3). Эти параметры определяют стратегию эволюции и позволяют экспериментировать с различными настройками, чтобы оценить их влияние на результат. Алгоритм реализован по островной модели с четырьмя независимыми популяциями, каждая из которых инициализируется своим методом: северо-западного угла, минимального элемента, решателем HiGHS или случайной генерацией – рисунок 4.

После запуска алгоритма пользователь получает информацию о ходе эволюции: номер текущего поколения, лучшее значение стоимости и общее

время работы – рисунок 4. Одновременно отображается текущий лучший транспортный план и таблица, сравнивающая стоимость решений на каждом из островов. Это даёт возможность отслеживать сходимость островов.

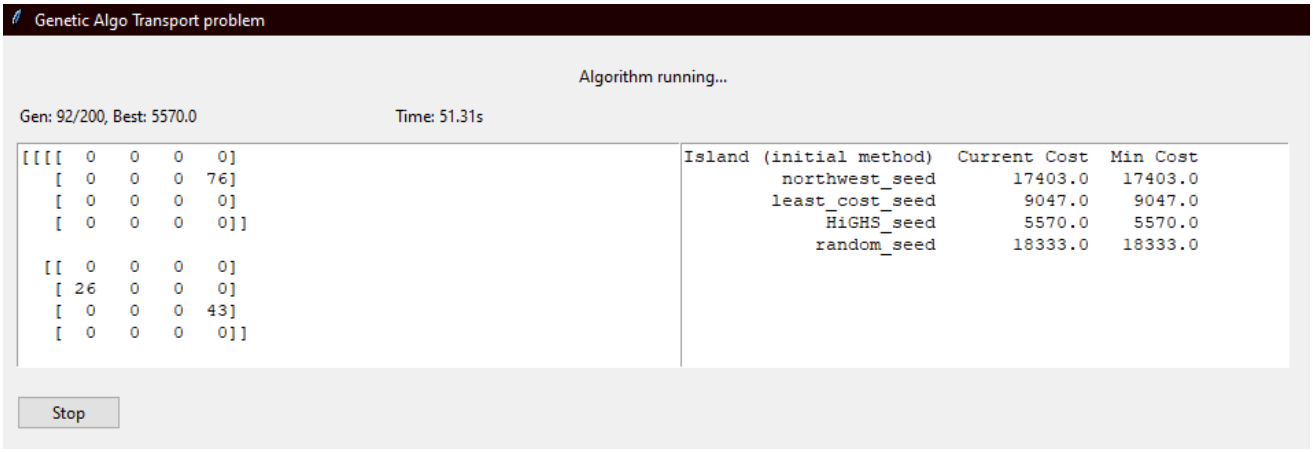


Рисунок 4 – Работа алгоритма

По завершении оптимизации программа формирует итоговое окно с расширенной аналитикой – рисунок 5. Отображается итоговая стоимость, финальный транспортный план, проверка соблюдения ограничений по каждой оси (в том числе недогруз и превышение), а также сравнение с результатами, полученными начальной генерацией с использованием классических методов. Такая интеграция встроенного анализа делает возможным прямое сравнение ГА с классическими подходами без необходимости запускать их отдельно.

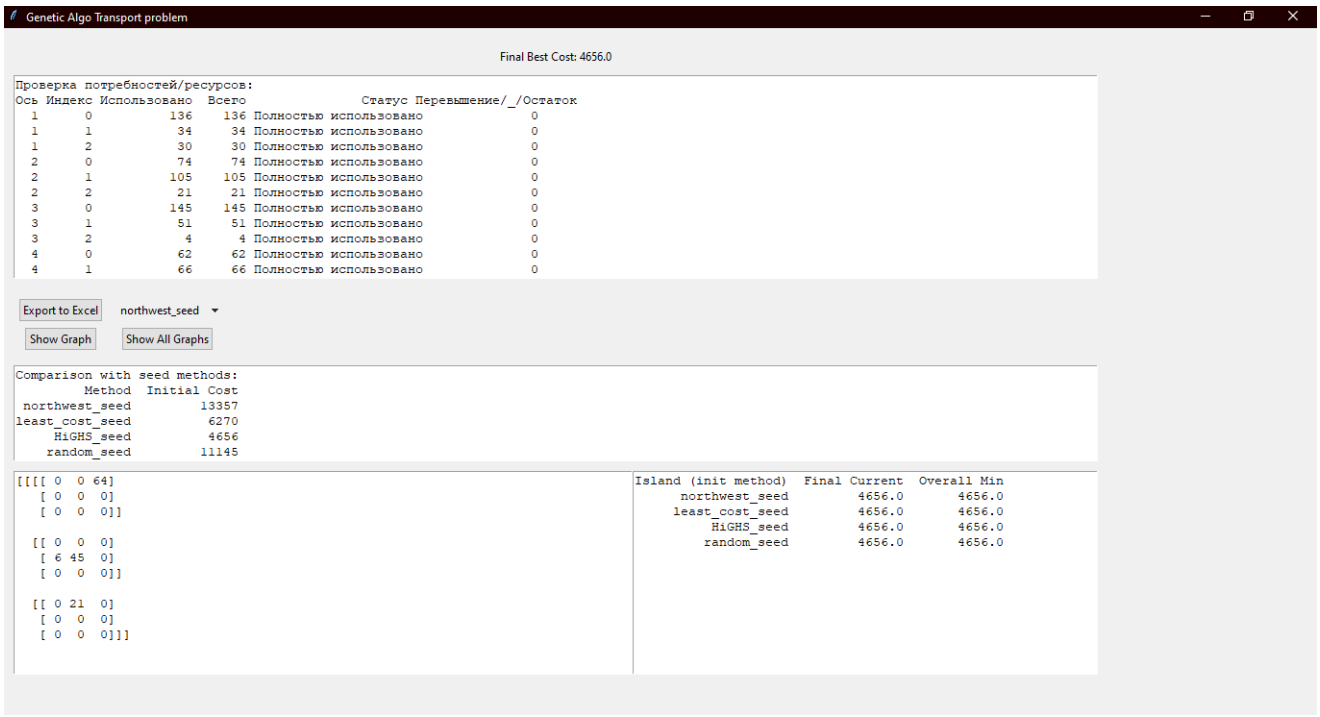


Рисунок 5 – Окно с выходными данными и аналитикой

В дополнение к текстовой информации пользователю доступны графики сходимости для каждого острова отдельно – рисунок 6, рисунок 7, или для всех одновременно – рисунок 8. Предусмотрена возможность экспорта полученного плана в Excel – рисунок 9, что облегчает последующую работу с результатами и их включение в отчёты или внешние системы. Таким образом, интерфейс выполняет не только вспомогательную роль, но и служит полноценной аналитической панелью, позволяя как проводить исследования, так и использовать продукт в прикладной логистике.

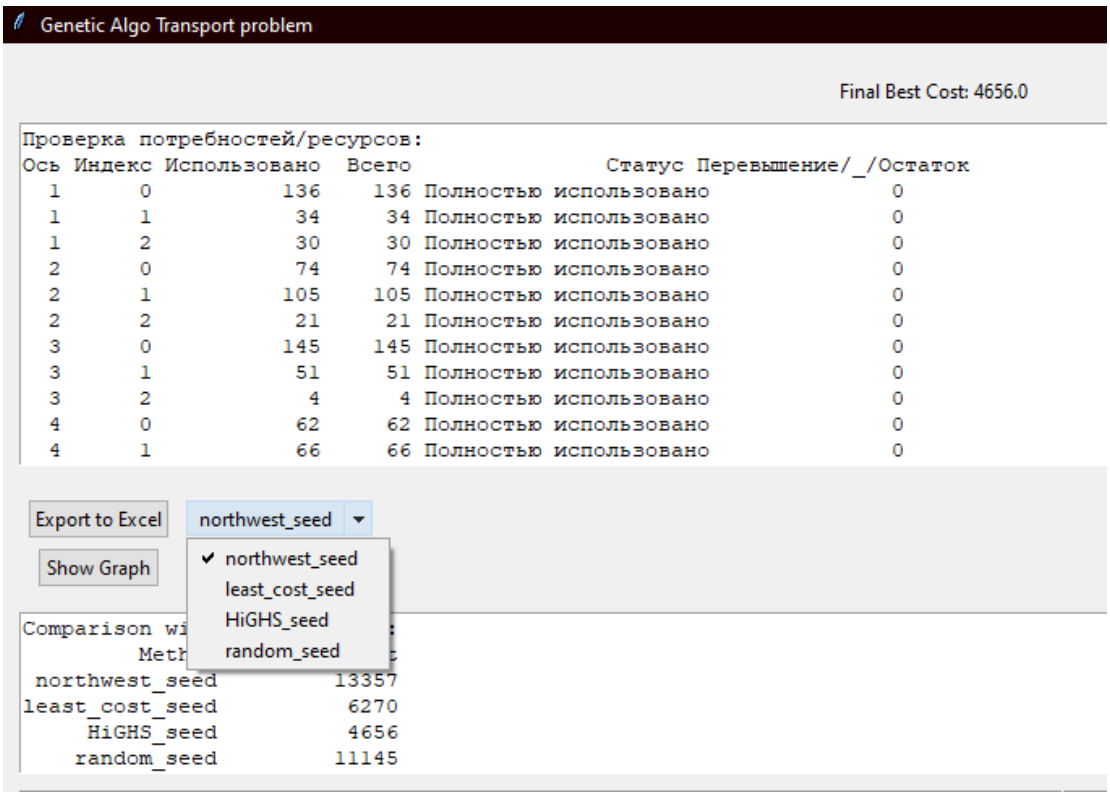


Рисунок 6 – Выбор графика для отображения

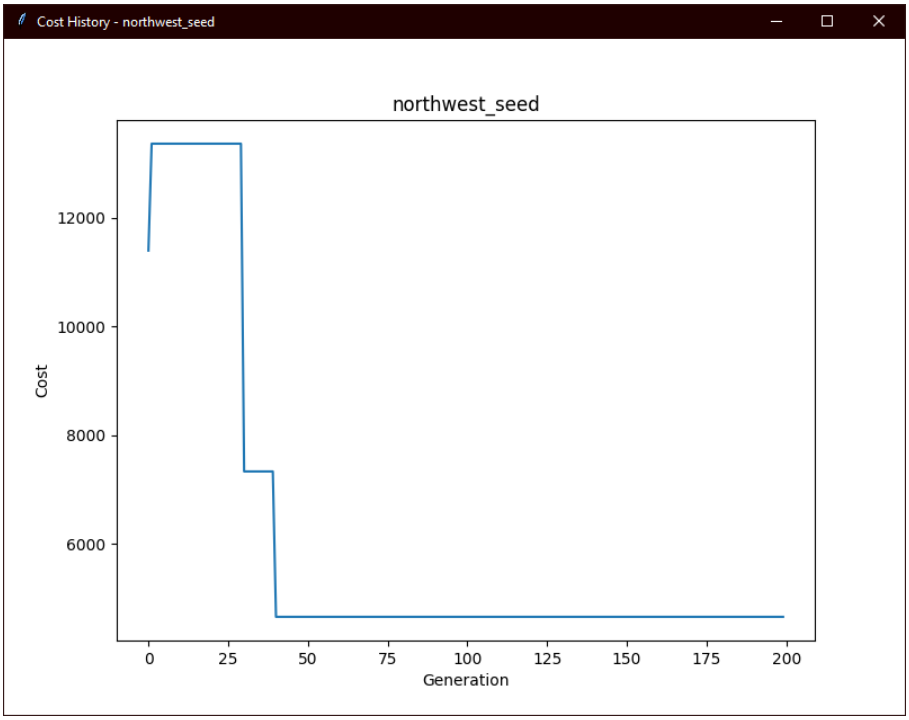


Рисунок 7 – Графики минимальной стоимости для метода СЗУ

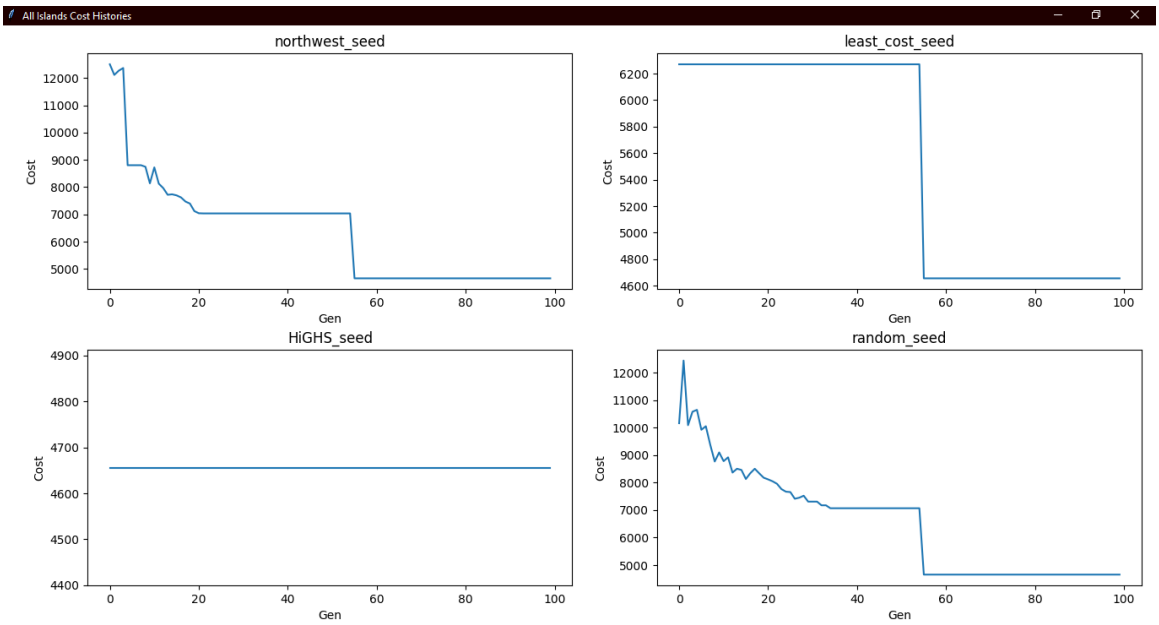
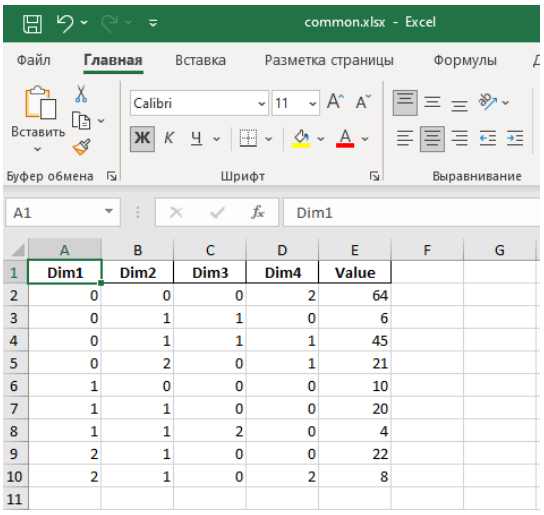


Рисунок 8 – Графики минимальной стоимости по островам для сбалансированных данных



	A	B	C	D	E
1	Dim1	Dim2	Dim3	Dim4	Value
2	0	0	0	2	64
3	0	1	1	0	6
4	0	1	1	1	45
5	0	2	0	1	21
6	1	0	0	0	10
7	1	1	0	0	20
8	1	1	2	0	4
9	2	1	0	0	22
10	2	1	0	2	8
11					

Рисунок 9 – Итоговый план перевозок, экспортированный в эксель

Разработанный GUI охватывает все этапы работы с задачей: от загрузки и настройки до анализа и сравнения результатов. Это делает программу не только инструментом для проведения вычислений, но и удобным средством исследования поведения алгоритма при различных параметрах и структурах данных.

4.3 Тестирование на сбалансированных данных

На втором и третьем этапе тестирование проходило в автоматическом режиме через специально разработанную под это программу (приложение Б). В рамках тестирования оценивалась работа генетического алгоритма на задачах, в которых сумма ресурсов и потребностей по каждой оси совпадает. Такие задачи представляют собой классическую сбалансированную транспортную постановку и позволяют проверить как корректность реализации, так и устойчивость алгоритма к различным структурам данных.

Для тестирования программной системы и оценки устойчивости алгоритма были задействованы задачи различной размерности – от классических двумерных до обобщённых семимерных. На начальном этапе реализация проходила верификацию на примерах, заимствованных из профильных учебников и пособий по транспортным задачам. Результаты, полученные программой, полностью совпали с эталонными решениями, что подтвердило корректность как классических методов, так и структуры генетического алгоритма. После этого было принято решение использовать разработанный мной генератор задач (приложение В), формирующий как сбалансированные, так и несбалансированные постановки произвольной размерности – как в двумерной, так и в многоиндексной форме. Такой подход дал возможность более полно охватить потенциальные сценарии, встречающиеся в реальных бизнес-процессах, включая те, где отсутствует чёткое соответствие между спросом и предложением или присутствуют дополнительные уровни логистических связей.

Для обеспечения сопоставимости результатов во всех тестах использовался единый набор параметров генетического алгоритма: размер популяции – 100 особей, число поколений – 100, вероятность мутации – 2 %, количество островов – четыре. Эти параметры были выбраны на основе предварительного анализа как обеспечивающие стабильную работу алгоритма. Несмотря на то что оптимальные параметры следовало бы

адаптировать под конкретную задачу, для целей тестирования были использованы единые настройки, демонстрирующие удовлетворительные результаты на задачах всех размерностей, даже если это сопровождалось избыточным числом поколений для задач меньшего масштаба. Формирование начальной популяции на каждом острове осуществлялось с использованием одного из следующих методов: метода северо-западного угла, метода минимального элемента, решателя HiGHS и случайного генератора. Такой подход обеспечивал возможность сопоставления эффективности различных стратегий инициализации в рамках одного запуска алгоритма.

На протяжении выполнения алгоритма собирались ключевые показатели: итоговая стоимость, количество поколений до стабилизации, время выполнения и количество допустимых решений в популяции. Кроме того, после завершения каждого эксперимента осуществлялась проверка соблюдения ограничений по каждой из осей, что позволяло зафиксировать случаи нарушения условий задачи.

Для каждого запуска визуализировалась динамика функции приспособленности по поколениям – как для отдельных островов, так и в совокупности. Это позволило оценить не только финальное качество решений, но и скорость сходимости. Примеры таких графиков представлены на рисунке 8. Во всех задачах наблюдались плавное обучение и резкие скачки в точках обмена решениями между островами, что говорит о корректной работе островной модели, операторов отбора, кроссовера и мутации.

Анализ финальных решений показал, что во всех тестах удавалось получить допустимый план без нарушений ограничений. При этом итоговая стоимость, полученная с помощью генетического алгоритма, либо совпадала, либо превосходила по качеству (то есть была ниже) те решения, которые формировались с помощью отдельных seed-методов. Таблица 3 ниже иллюстрирует сравнение средних итоговых стоимостей для всех примеров, а также демонстрирует, какой из методов инициализации оказался наиболее результативным в рамках каждого острова.

Таблица 3 – Сравнительный анализ качства решений

Характеристика	Метод северо-западного угла	Метод минимального элемента	Промышленный решатель HiGHS	Генетический алгоритм
Среднее значение решения	6900	3432	2573	2963
Среднее отставание от оптимума (ед.)	4328	860	0	390
Среднее отставание от оптимума (%)	168	33	0	15

Следует отметить, что время выполнения всех вычислений составляло от одной до трёх минут в зависимости от размерности задач, варьирующейся от двумерной (Таблица 4) до семимерной (Таблица 5). Это подтверждает применимость разработанной системы в реальных условиях, где важны не только точность, но и быстродействие. При этом адаптивная функция приспособленности, учитывающая штрафы за недогруз и превышение, не мешала эффективной сходимости алгоритма, а наоборот, способствовала стабилизации результатов.

Таблица 4 – Среднее отставание от оптимума при тестировании двумерных задач

Методы решения	Разменость	Среднее отставание от оптимума (ед.)	Среднее отставание от оптимума (%)	Лучшее решение (ед.)	Худшее решение (ед.)
Метод северо-западного угла	2d	32	1	7452	
Метод минимального элемента		682	17	6269	
Промышленный решатель HiGHS		0	0	5736	
Генетический алгоритм		0	0	5736	5736

Таблица 5 – Среднее отставание от оптимума при тестировании семимерных задач

Методы решения	Разменость	Среднее отставание от оптимума (ед.)	Среднее отставание от оптимума (%)	Лучшее решение (ед.)	Худшее решение (ед.)
Метод северо-западного угла	7d	4526	824	5065	
Метод минимального элемента		1698	309	2247	
Промышленный решатель HiGHS		0	0	549	
Генетический алгоритм		849	155	549	2247

Таким образом, тестирование на сбалансированных данных подтвердило как корректность реализации, так и высокую эффективность предложенного подхода. Алгоритм устойчив к размерности задачи, стабильно находит допустимые решения, а благодаря встроенному сравнению с базовыми методами позволяет наглядно оценить его преимущества.

4.4 Тестирование на несбалансированных данных

На следующем этапе была проведена серия экспериментов с использованием несбалансированных постановок транспортной задачи, при которых сумма предложений по одной или нескольким осям не совпадает с суммой потребностей. Подобные ситуации часто встречаются в практических логистических задачах, обусловленных наличием избыточных ресурсов, недогрузов либо неравномерного распределения спроса и предложения. Для обработки таких случаев применялась та же программная реализация генетического алгоритма, без необходимости внесения изменений в структуру кода или ручного переключения режимов работы.

Тестирование проводилось на наборах задач той же размерности, что и в предыдущем разделе, – от двумерных до семимерных. Использовались

идентичные параметры алгоритма, за исключением увеличенного числа поколений: для повышения вероятности получения качественных решений параметр был увеличен вдвое и составлял 200 поколений. Основное отличие заключалось в изменённой формулировке функции приспособленности, в которой учитывались штрафные значения за недопоставку или превышение предложения по каждой из осей. Это позволяло алгоритму эффективно различать допустимые и недопустимые решения, корректно направляя процесс эволюции.

В ходе экспериментов регистрировались значения итоговой целевой функции, количество нарушений ограничений, степень перераспределения ресурсов, а также доля допустимых особей в популяции на каждом поколении. Генетический алгоритм продемонстрировал устойчивое поведение: на начальных этапах эволюции преобладали частично допустимые решения, однако с увеличением числа поколений наблюдалось последовательное сокращение их доли и рост числа особей, соответствующих ограничениям. Это подтверждает корректность механизма фильтрации и способность алгоритма адаптироваться к сложным входным данным.

Дополнительно было проведено сравнение с классическими методами. Анализ графиков (рисунок 10) показал, что при использовании методов северо-западного угла и минимального элемента на начальных генерациях генерировались решения, не удовлетворяющие ограничениям, в результате чего наблюдался рост значения целевой функции. После появления допустимых решений стоимость начинала снижаться. Примечательно, что даже при использовании решателя HiGHS (график lp_method) в случае несбалансированных задач на начальных этапах не удется получить оптимальные решения, что подчёркивает сложность подобных постановок и целесообразность применения адаптивных эвристических подходов.

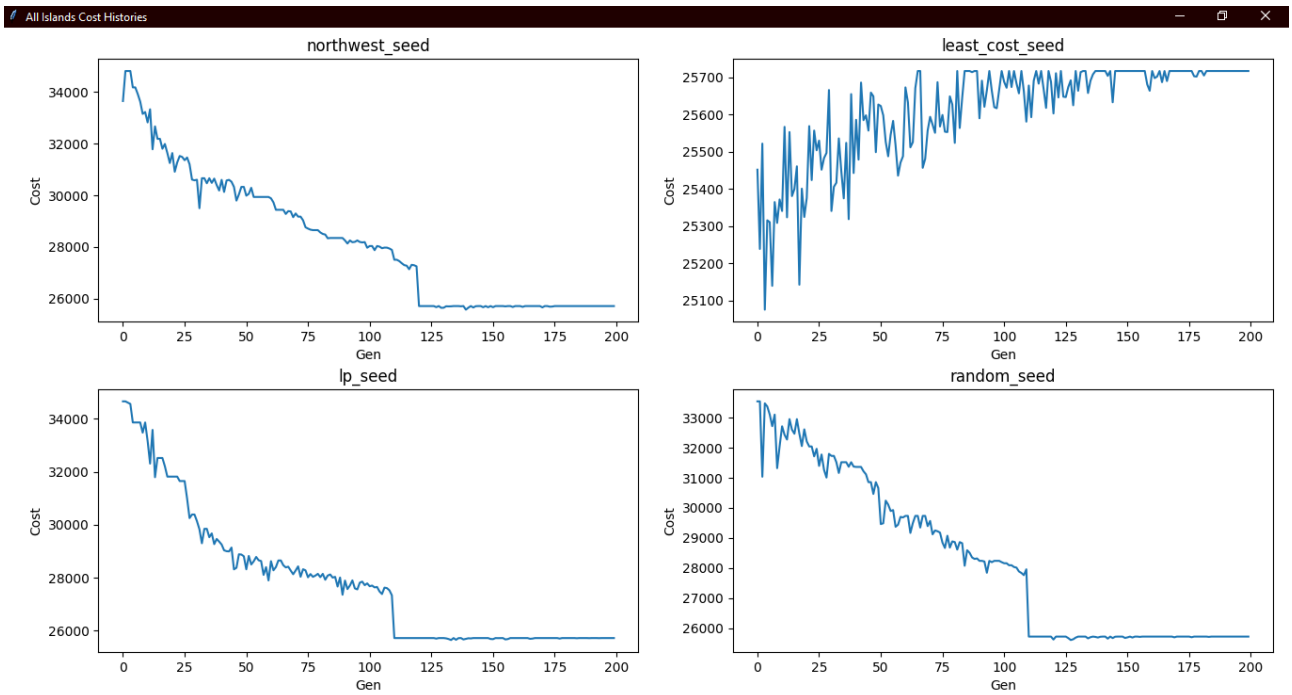


Рисунок 10 – Графики минимальной стоимости по островам для несбалансированных данных

Финальные планы, полученные на несбалансированных данных, отражали точный учёт логистических ограничений. Проверка соответствия требованиям проводилась автоматически – в пользовательском интерфейсе отображались отклонения по каждой оси, что позволяло сразу проанализировать "слабые места" полученного плана – рисунок 11.

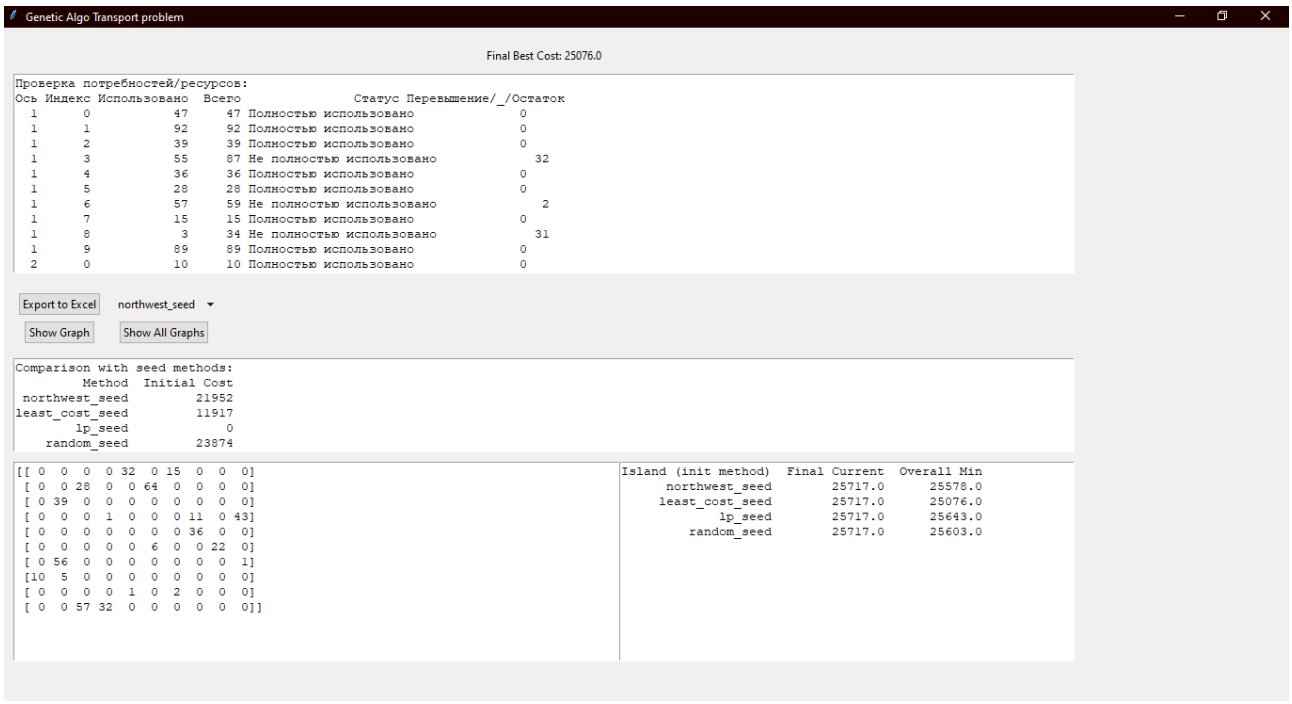


Рисунок 11 – Вывод алгоритма на несбалансированных данных

Визуализация решений, графики сходимости и сравнение с результатами, полученными базовыми методами инициализации (рисунок 10, рисунок 11), показали, что использование генетического алгоритма остаётся эффективным и в условиях несбалансированных систем. Несмотря на возможное отсутствие точного "идеального" решения, метод способен адаптироваться к структуре задачи и выдавать планы, которые с практической точки зрения являются экономически и логистически целесообразными.

Проведённые эксперименты подтверждают, что предложенный подход применим не только для строго формализованных задач, но и для более реалистичных, "шумных" сценариев с избыточными или недостаточными ресурсами. Это расширяет спектр применения разработанной системы и делает её подходящей для задач распределения в условиях неопределённости.

Выводы

В рамках выполнения выпускной квалификационной работы были реализованы все этапы по созданию и исследованию программной системы для решения транспортных задач произвольной размерности. Основное внимание было уделено многоиндексным и несбалансированным постановкам, наиболее характерным для прикладных логистических сценариев. Работа включала как теоретическое обоснование выбранных подходов, так и практическую реализацию универсального программного продукта.

Проведён анализ классических, эвристических и метаэвристических методов решения транспортных задач, их применимость к задачам с высокой размерностью и нарушением баланса. Обоснован выбор генетического алгоритма в качестве основного метода.

Разработано единое универсальное представление транспортной задачи, поддерживающее любую размерность и способное описывать как классическую, так и обобщённую форму с несколькими индексами. Предложенная структура хранения обеспечивает совместимость со всеми методами генерации и анализа решений.

Реализован генетический алгоритм с островной моделью, адаптированный для многоиндексных задач. Особое внимание было уделено корректному кодированию хромосом, реализации функции приспособленности с адаптивными штрафами, а также операторам кроссовера и мутации, обеспечивающим допустимость потомков.

Разработана и протестирована система генерации начальной популяции с использованием нескольких методов (северо-западный угол, минимальный элемент, решатель HiGHS, случайная генерация), что позволило повысить разнообразие особей и ускорить сходимость алгоритма.

Разработан графический интерфейс пользователя, позволяющий работать с задачами произвольной размерности, задавать параметры

алгоритма, наблюдать ход выполнения, сравнивать результаты разных методов и экспортировать решения в Excel. Интерфейс реализует полную визуализацию процесса, включая ограничения, структуру решений и графики эволюции.

Проведено комплексное тестирование на сгенерированных задачах различной размерности и степени сбалансированности. Для этого был разработан собственный генератор задач, учитывающий как учебные, так и бизнес-ориентированные сценарии. Эксперименты показали, что алгоритм демонстрирует устойчивую сходимость, способность формировать допустимые планы, а также превосходство над классическими методами по качеству решения.

Таким образом, поставленные в работе цели были достигнуты, а разработанная система продемонстрировала свою применимость в широком классе задач транспортной логистики и оптимизации распределения ресурсов.

Заключение

В данной выпускной квалификационной работе была рассмотрена задача разработки и исследования универсальной программной системы для решения транспортных задач, включая их многоиндексные и несбалансированные варианты, актуальные для современной логистики.

На основе анализа существующих методов было обосновано применение генетического алгоритма с островной моделью, обладающего необходимой гибкостью и способностью справляться с задачами произвольной структуры. Были учтены особенности реализации таких алгоритмов в условиях высокой размерности, включая сложности кодирования решений, ограничения по допустимости, а также вопросы масштабируемости.

Разработанная система реализует единый формат хранения и обработки данных, поддерживает несколько методов построения начальных решений и включает встроенные средства визуализации. Благодаря графическому интерфейсу пользователь может гибко задавать параметры задачи и алгоритма, получать развернутые результаты и экспортировать их в удобном формате.

Проведённые эксперименты подтвердили корректность работы реализованных алгоритмов, их устойчивость и применимость к практическим задачам. Сравнение с классическими методами показало преимущества предложенного подхода в задачах, выходящих за пределы стандартных учебных примеров.

Полученные результаты могут быть использованы при дальнейшем развитии программных средств поддержки принятия решений в области логистики, а также в учебных целях при изучении методов оптимизации и эволюционных алгоритмов. Разработанная система обладает достаточным потенциалом для расширения, включая интеграцию дополнительных критериев, ограничений и методов оптимизации.

Список используемой литературы и используемых источников

1. Ахмедов И. У., Хусаинов Д. И. Многоиндексные транспортные задачи: теория и приложения. Казань : КФУ, 2018. 148 с.
2. Баженов А. Ю., Кузнецов Д. В. Алгоритмы и методы оптимизации. М. : БХВ-Петербург, 2020. 336 с.
3. Голодов В. И. Методы оптимизации: транспортные задачи и их обобщения. СПб. : Питер, 2019. 384 с.
4. Голдберг Д. Э. Генетические алгоритмы в поиске, оптимизации и машинном обучении. М. : Мир, 1989. 352 с.
5. Деб К. Multi-objective optimization using evolutionary algorithms. Chichester : Wiley, 2001. 518 p.
6. Журавлёв Ю. И., Ковальчук В. И. Методы оптимизации и их приложения. М. : Наука, 2010. 412 с.
7. Коэлло К. А., Ван Вретен Дж., Ламонт Г. Б. Эволюционные алгоритмы для многокритериальной оптимизации. М. : Логос, 2007. 480 с.
8. Лапин В. В. Транспортные задачи: методы решения и примеры. М. : Финансы и статистика, 2012. 272 с.
9. Michalewicz Z. Genetic algorithms + data structures = evolution programs. Berlin : Springer, 1996. 471 p.
10. Meng Q., Wu J., Ellisy J., Kennedy P. J. Dynamic island model based on spectral clustering in genetic algorithm // arXiv. 2018. URL: https://arxiv.org/abs/1801.01620 (дата обращения: 23.02.2025).
11. McKinney W. Python for data analysis. 3rd ed. Sebastopol : O’Reilly Media, 2022. 550 p.
12. Harris C. R., Millman K. J., van der Walt S. J. et al. Array programming with NumPy // Nature. 2020. Vol. 585. P. 357–362.
13. Gendreau M., Potvin J.-Y. (eds.) Handbook of metaheuristics. 3rd ed. New York : Springer, 2019. 604 p.
14. Gleixner A. M., Steffy D. E., Wright S. J. The HiGHS linear

optimization software // arXiv. 2020. URL: https://arxiv.org/abs/2002.04065 (дата обращения: 28.03.2025).

15. Holland J. H. Adaptation in natural and artificial systems. Cambridge : MIT Press, 1975. 228 p.

16. Hunter J. D. Matplotlib: a 2D graphics environment // Computing in Science & Engineering. 2007. Vol. 9(3). P. 90–95.

17. Поляков В. А. Методы и алгоритмы оптимизации в задачах логистики. М. : Логистика, 2015. 320 с.

18. Рутковская М., Пильчевский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. М. : Горячая линия – Телеком, 2006. 452 с.

19. Simon D. Evolutionary optimization algorithms. Hoboken : Wiley, 2013. 512 p.

20. Taha H. A. Operations research: an introduction. 10th ed. Boston : Pearson, 2017. 848 p.

21. Vicary J. Implementing the HiGHS linear constraint solver in Rust. Cambridge : University of Cambridge, 2022. URL: https://github.com/jlcv/highs-rs (дата обращения: 28.03.2025).

22. Simoncini D., Collard P., Verel S., Clergue M. From cells to islands: an unified model of cellular parallel genetic algorithms // arXiv. 2008. URL: https://arxiv.org/abs/0803.4248 (дата обращения: 23.02.2025).

23. Хаханова Н. Н. Исследование операций: учебное пособие. М. : Инфра-М, 2021. 256 с.

Приложение А

Код программы для решения транспортной задачи

```
import json
import random
import threading
import time
import numpy as np
from copy import deepcopy
from scipy.optimize import linprog
import pandas as pd
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
import os
import sys
sys.stdout.reconfigure(encoding="utf-8")
def read_data_from_file(path):
    with open(path, 'r', encoding='utf-8') as f:
        data = json.load(f)
    constraints = [np.array(m) for m in data['margins']]
    costs = np.array(data['costs'])
    return constraints, costs
def northwest_seed(constraints, costs):
    rem, plan = [c.copy() for c in constraints], np.zeros(costs.shape, int)
    shape, idx = costs.shape, [0]*costs.ndim
    while all(idx[d] < shape[d] for d in range(costs.ndim)):
        avail = min(rem[d][idx[d]] for d in range(costs.ndim))
        plan[tuple(idx)] = avail
        for d in range(costs.ndim): rem[d][idx[d]] -= avail
        for d in range(costs.ndim):
            if rem[d][idx[d]] == 0:
                idx[d] += 1
                break
    return plan
def least_cost_seed(constraints, costs):
    rem, plan = [c.copy() for c in constraints], np.zeros(costs.shape, int)
    cells = list(np.ndindex(costs.shape)); cells.sort(key=lambda i: costs[i])
    for i in cells:
        avail = min(rem[d][i[d]] for d in range(costs.ndim))
        if avail > 0:
            plan[i] = avail
            for d in range(costs.ndim): rem[d][i[d]] -= avail
    return plan
def HiGHS_seed(constraints, costs):
    sizes, c = list(costs.shape), costs.flatten()
    A_eq, b_eq = [], []
    for dim, sz in enumerate(sizes):
        for pos in range(sz):
            row = np.zeros(costs.size)
            for idx, coord in enumerate(np.ndindex(*sizes)):
                if coord[dim] == pos:
                    row[idx] = 1
            A_eq.append(row); b_eq.append(constraints[dim][pos])
    res = linprog(c, A_eq=A_eq, b_eq=b_eq, bounds=[(0, None)]*c.size, method='highs')
    plan = res.x.reshape(sizes) if res.success else np.zeros(sizes)
    return np.round(plan).astype(int)
def random_seed(constraints, costs):
```

```
    rem, plan = [c.copy() for c in constraints], np.zeros(costs.shape, int)
    cells = list(np.ndindex(costs.shape)); random.shuffle(cells)
    for i in cells:
        avail = min(rem[d][i[d]] for d in range(costs.ndim))
        if avail > 0:
            x = random.randint(0, avail); plan[i] = x
            for d in range(costs.ndim): rem[d][i[d]] -= x
    return plan
def violation_summary(plan, constraints):
    total_over = 0
    total_under = 0
    dims = plan.ndim
    axes = list(range(dims))
    for axis in axes:
        proj = plan.sum(axis=tuple(a for a in axes if a != axis))
        diff = proj - constraints[axis]
        total_over  += diff.clip(min=0).sum()
        total_under += (-diff).clip(min=0).sum()
    return int(total_over), int(total_under)
def is_valid(plan, constraints):
    for ax in range(plan.ndim):
        proj = plan.sum(axis=tuple(i for i in range(plan.ndim) if i != ax))
        if np.any(proj > constraints[ax]): return False
    return True
def fitness(plan, costs, constraints, base_over=50, base_under=50, gen=0, total_gens=1):
    over = base_over * (1 + gen/total_gens)
    under = base_under * (1 + gen/total_gens)
    val = np.sum(plan * costs)
    for ax in range(plan.ndim):
        proj = plan.sum(axis=tuple(i for i in range(plan.ndim) if i != ax))
        diff = proj - constraints[ax]
        val += over * np.sum(diff.clip(min=0))
        val += under * np.sum((-diff).clip(min=0))
    return val
def local_hill_climb(plan, costs, constraints, steps=100):
    best, bc = plan.copy(), fitness(plan, costs, constraints)
    for _ in range(steps):
        src = tuple(random.randrange(s) for s in plan.shape)
        dst = tuple(random.randrange(s) for s in plan.shape)
        if best[src] > 0:
            cand = best.copy(); cand[src] -= 1; cand[dst] += 1
            if is_valid(cand, constraints):
                fc = fitness(cand, costs, constraints)
                if fc < bc: best, bc = cand, fc
    return best
def simulated_annealing(plan, costs, constraints, T0=1.0, alpha=0.99, steps=100):
    cur, cc = plan.copy(), fitness(plan,costs,constraints)
    best, bc = cur.copy(), cc; T=T0
    for _ in range(steps):
        idx = tuple(random.randrange(s) for s in plan.shape)
        if cur[idx] > 0:
            cand = cur.copy(); cand[idx] -= 1
            jdx = tuple(random.randrange(s) for s in plan.shape); cand[jdx] += 1
            if is_valid(cand,constraints):
                fc = fitness(cand,costs,constraints); dE = fc-cc
                if dE<0 or random.random()<np.exp(-dE/T): cur,cc = cand,fc
                if fc<bc: best,bc = cand.copy(),fc
        T *= alpha
    return best
def variable_neighborhood(plan, constraints, costs, mutation_rate):
    r = random.choice([0,1,2])
    if r == 0: return mutate(plan, mutation_rate)
```

```
    if r == 1: return local_hill_climb(plan, costs, constraints)
    return simulated_annealing(plan, costs, constraints)
def mutate(plan, rate):
    for idx in np.ndindex(*plan.shape):
        if random.random() < rate: plan[idx] = max(0, plan[idx] + random.randint(-1,1))
    return plan
def multi_start(constraints, costs, trials=50):
    best, bc = None, float('inf')
    for _ in range(trials):
        p = random_seed(constraints,costs)
        p = local_hill_climb(p,costs,constraints)
        c = fitness(p,costs,constraints)
        if c < bc: best,bc = p.copy(),c
    return best
def tournament_select(pop, costs, constraints, gen, total_gens, k=3):
    picks = random.sample(pop, k)
    return min(picks, key=lambda p: fitness(p,costs,constraints,gen=gen,total_gens=total_gens))
def crossover(a, b):
    mask = np.random.rand(*a.shape) < 0.5
    return np.where(mask, a, b)
def genetic_algorithm_islands(constraints, costs,
                              pop_size=100, generations=100,
                              islands=4, seed_ratio=0.25,
                              mutation_rate=0.02, retain_frac=0.5, elitism=5,
                              iso_ratio=0.5, comm_ratio=0.05,
                              update_gui=None, stop_event=None):
    iso_gens = int(generations * iso_ratio)
    comm_int = max(1, int(generations * comm_ratio))
    best0 = multi_start(constraints, costs, 30)
    seeds = [northwest_seed, least_cost_seed, HiGHS_seed, random_seed]
    archi = []
    for isl in range(islands):
        pop = [best0.copy()]
        m = seeds[isl % len(seeds)]; n_seed = int((pop_size-1)*seed_ratio)
        for _ in range(n_seed): pop.append(m(constraints,costs))
        for _ in range(pop_size-1-n_seed): pop.append(random_seed(constraints,costs))
        archi.append(pop)
    logs = [[None]*generations for _ in range(islands)]
    global_best, global_cost = None, float('inf')
    for gen in range(1, generations+1):
        if stop_event and stop_event.is_set(): break
        for isl in range(islands):
            pop = archi[isl]
            valid = [p for p in pop if is_valid(p,constraints)]
            need = int(pop_size*retain_frac) - len(valid)
            for _ in range(max(0, need)): valid.append(random_seed(constraints,costs))
            valid.sort(key=lambda p: fitness(p,costs,constraints,gen, generations))
            elites = valid[:elitism]; children = []
            while len(children) < pop_size-elitism:
                p1 = tournament_select(valid,costs,constraints,gen,generations)
                p2 = tournament_select(valid,costs,constraints,gen,generations)
                c = crossover(p1,p2)
                children.append(variable_neighborhood(c,constraints,costs,mutation_rate))
            archi[isl] = elites + children
            bl = min(archi[isl], key=lambda p: fitness(p,costs,constraints,gen,generations))
            bc = fitness(bl,costs,constraints,gen,generations)
            logs[isl][gen-1] = bc
            over, under = violation_summary(bl, constraints)
            if not (over > 0 and under > 0) and bc < global_cost:
                global_cost, global_best = bc, bl.copy()
        if gen>iso_gens and (gen-iso_gens)%comm_int==0:
            migrants=[]
```

```
            for pop in archi:
                pop.sort(key=lambda p: fitness(p,costs,constraints,gen,generations))
                migrants.extend(deepcopy(pop[:elitism]))
            for isl in range(islands):
                archi[isl].sort(key=lambda p: fitness(p,costs,constraints,gen,generations), reverse=True)
                for i in range(elitism): archi[isl][i] = random.choice(migrants)
        if update_gui: update_gui(gen, logs, global_best, global_cost)
    return global_best, global_cost, logs
def export_plan_to_excel(plan, filename):
    shape = plan.shape
    indices = list(np.ndindex(shape))
    data = [list(idx) + [plan[idx]] for idx in indices if plan[idx] > 0]
    cols = [f"Dim{i+1}" for i in range(plan.ndim)] + ["Value"]
    df = pd.DataFrame(data, columns=cols)
    df.to_excel(filename, index=False)
def run_gui():
    root = tk.Tk()
    root.title("Genetic Algo Transport problem")
    root.state("zoomed")
    constraints, costs = None, None
    seed_methods = [northwest_seed, least_cost_seed, HiGHS_seed, random_seed]
    stop_event = threading.Event()
    def on_closing():
        if messagebox.askokcancel("Выход", "Вы действительно хотите выйти?"):
            stop_event.set()
            root.destroy()
            os._exit(0)
    root.protocol("WM_DELETE_WINDOW", on_closing)
    main_frame = ttk.Frame(root, padding=10)
    main_frame.pack(fill=tk.BOTH, expand=True)
    constraints, costs = None, None
    seed_methods = [northwest_seed, least_cost_seed, HiGHS_seed, random_seed]
    stop_event = threading.Event()
    def add_copy_menu(text_widget):
        menu = tk.Menu(text_widget, tearoff=0)
        menu.add_command(label="Copy", command=lambda: text_widget.event_generate("<<Copy>>"))
        text_widget.bind("<Button-3>", lambda e: menu.tk_popup(e.x_root, e.y_root))
        text_widget.bind("<Control-c>", lambda e: text_widget.event_generate("<<Copy>>"))
    def load_file():
        nonlocal constraints, costs
        fn = filedialog.askopenfilename(filetypes=[('JSON','*.json')])
        if not fn:
            return
        constraints, costs = read_data_from_file(fn)
        ttk.Label(main_frame, text=f"Loaded: {fn}") \
            .grid(row=0, column=3, columnspan=4, pady=(0, 5), sticky='w')
        matrix_text = tk.Text(main_frame, height=30, width=100)
        matrix_text.grid(row=4, column=0, columnspan=4, pady=(5, 10), sticky='nsew')
        add_copy_menu(matrix_text)
        matrix_text.insert(tk.END, "Constraints (margins):\n")
        for i, vec in enumerate(constraints, start=1):
            matrix_text.insert(tk.END, f" dim {i}: {vec.tolist()}\n")
        matrix_text.insert(tk.END, "\nCosts matrix:\n")
        def print_ndarray(arr, prefix=""):
            arr = np.array(arr)
            if arr.ndim <= 2:
                df = pd.DataFrame(arr)
                matrix_text.insert(tk.END, f"{prefix}\n{df.to_string(index=False)}\n\n")
            else:
                for idx, subarr in enumerate(arr):
                    print_ndarray(subarr, prefix=f"{prefix}[{idx}]")
        print_ndarray(costs, prefix="Matrix_")
```

```
ttk.Button(main_frame, text='Load JSON', command=load_file) \
   .grid(row=0, column=0, pady=5, sticky='w')
def start_algorithm():
   nonlocal stop_event
   try:
      psz = int(pop_entry.get())
      gsz = int(gen_entry.get())
      iso_ratio = float(iso_ratio_entry.get())
      mutation_rate = float(mutation_entry.get())
      isz = 4
   except ValueError:
      messagebox.showerror("Error", "Invalid parameters")
      return
   if constraints is None or costs is None:
      messagebox.showerror("Error", "Load data first")
      return
   for widget in main_frame.winfo_children():
      widget.destroy()
   ttk.Label(main_frame, text="Algorithm running...").grid(row=0, column=0, columnspan=4, pady=10)
   gen_label = ttk.Label(main_frame, text="Gen: 0/0")
   gen_label.grid(row=1, column=0, sticky='w')
   time_label = ttk.Label(main_frame, text="Time: 0.00s")
   time_label.grid(row=1, column=1, sticky='w')
   plan_text = tk.Text(main_frame, height=10, width=60)
   plan_text.grid(row=2, column=0, columnspan=2, pady=10, sticky='nsew')
   add_copy_menu(plan_text)
   cost_text = tk.Text(main_frame, height=10, width=55)
   cost_text.grid(row=2, column=2, columnspan=2, pady=10, sticky='nsew')
   add_copy_menu(cost_text)
   stop_event.clear()
   stop_btn = ttk.Button(main_frame, text="Stop", command=lambda: stop_event.set())
   stop_btn.grid(row=3, column=0, pady=10, sticky='w')
   start_time = time.time()
   def update_generation(gen, logs, best_plan, best_cost):
      elapsed = time.time() - start_time
      gen_label.config(text=f"Gen: {gen}/{gsz}, Best: {best_cost:.1f}")
      time_label.config(text=f"Time: {elapsed:.2f}s")
      plan_text.delete('1.0', tk.END)
      plan_text.insert(tk.END, str(best_plan))
      current_costs = []
      min_costs = []
      for hist in logs:
         vals = [v for v in hist[:gen] if v is not None]
         if vals:
            current_costs.append(vals[-1])
            min_costs.append(min(vals))
         else:
            current_costs.append(float('nan'))
            min_costs.append(float('nan'))
      df = pd.DataFrame({
         'Island (initial method)': [seed_methods[i % len(seed_methods)].__name__ for i in range(len(logs))],
         'Current Cost': current_costs,
         'Min Cost':     min_costs
      })
      cost_text.delete('1.0', tk.END)
      cost_text.insert(tk.END, df.to_string(index=False))
   def genetic_thread():
      best, cost, logs = genetic_algorithm_islands(
         constraints, costs,
         pop_size=psz,
         generations=gsz,
         islands=isz,
```

```
            iso_ratio=iso_ratio,
            mutation_rate=mutation_rate,
            update_gui=update_generation,
            stop_event=stop_event
        )
        clean_logs = [[v for v in isl if v is not None] for isl in logs]
        show_results(best, cost, clean_logs)
    threading.Thread(target=genetic_thread, daemon=True).start()
def show_results(best_plan, best_cost, logs):
    for widget in main_frame.winfo_children(): widget.destroy()
    ttk.Label(main_frame, text=f"Final Best Cost: {best_cost:.1f}")\
        .grid(row=0, column=0, columnspan=4, pady=10)
    sat_info = []
    for ax in range(best_plan.ndim):
        proj = best_plan.sum(axis=tuple(i for i in range(best_plan.ndim) if i != ax))
        for idx, used in enumerate(proj):
            total = constraints[ax][idx]
            if used == total:
                status = "Полностью использовано"
                left = 0
            elif used < total:
                status = "Не полностью использовано"
                left = total - used
            else:
                status = "Превышение!"
                left = used - total
            sat_info.append((ax+1, idx, used, total, status, left))
    sat_txt = tk.Text(main_frame, height=13, width=100)
    sat_txt.grid(row=1, column=0, columnspan=4, pady=(0,10), sticky='nsew')
    add_copy_menu(sat_txt)
    sat_txt.insert(tk.END, "Проверка потребностей/ресурсов:\n")
    sat_txt.insert(tk.END, f"{'Ось':>3} {'Индекс':>6} {'Использовано':>12} "
                           f"{'Всего':>6} {'Статус':>20} {'Перевышение/_/Остаток':>14}\n")
    for ax, idx, used, total, status, left in sat_info:
        sat_txt.insert(tk.END,
            f"{ax:>3} {idx:>6} {used:>12} {total:>6} {status:>20} {left:>14}\n"
        )
    sat_txt.configure(state='disabled')
    init_results = []
    for method in seed_methods:
        plan0 = method(constraints, costs)
        cost0 = np.sum(plan0 * costs)
        init_results.append({
            'Method': method.__name__,
            'Initial Cost': cost0
        })
    df_init = pd.DataFrame(init_results)
    init_txt = tk.Text(main_frame, height=len(init_results)+2, width=50)
    init_txt.grid(row=5, column=0, columnspan=4, pady=(0,10), sticky='nsew')
    add_copy_menu(init_txt)
    init_txt.insert(tk.END, "Comparison with seed methods:\n")
    init_txt.insert(tk.END, df_init.to_string(index=False))
    init_txt.configure(state='disabled')
    txt = tk.Text(main_frame, height=13)
    txt.grid(row=6, column=0, columnspan=2, sticky='nsew')
    add_copy_menu(txt)
    txt.insert(tk.END, str(best_plan))
    cost_txt = tk.Text(main_frame, height=13, width=60)
    cost_txt.grid(row=6, column=2, columnspan=2, sticky='nsew')
    add_copy_menu(cost_txt)
    final_current = [logs[i][-1] for i in range(len(logs))]
    final_min     = [min(logs[i])    for i in range(len(logs))]
```

```
        df_final = pd.DataFrame({
            'Island (init method)': [seed_methods[i % len(seed_methods)].__name__ for i in range(len(logs))],
            'Final Current': final_current,
            'Overall Min':   final_min
        })
        cost_txt.insert(tk.END, df_final.to_string(index=False))
        btn_frame = ttk.Frame(main_frame)
        btn_frame.grid(row=2, column=0, columnspan=4, pady=10, sticky='w')
        def export():
            fn = filedialog.asksaveasfilename(defaultextension='.xlsx', filetypes=[('Excel','*.xlsx')])
            if not fn: return
            export_plan_to_excel(best_plan, fn)
            messagebox.showinfo('Export', 'Plan exported to ' + fn)
        exp_btn = ttk.Button(btn_frame, text="Export to Excel", command=export)
        exp_btn.grid(row=0, column=0, padx=5)
        options = [seed_methods[i % len(seed_methods)].__name__ for i in range(len(logs))]
        graph_var = tk.StringVar(); graph_var.set(options[0])
        graph_menu = ttk.OptionMenu(btn_frame, graph_var, options[0], *options)
        graph_menu.grid(row=0, column=1, padx=5)
        def show_graph():
            idx = options.index(graph_var.get())
            w = tk.Toplevel(root)
            w.title(f"Cost History - {options[idx]}")
            fig, ax = plt.subplots(figsize=(8,6))
            ax.plot(logs[idx])
            ax.set_title(f'{options[idx]}')
            ax.set_xlabel('Generation')
            ax.set_ylabel('Cost')
            c = FigureCanvasTkAgg(fig, master=w)
            c.draw(); c.get_tk_widget().pack(fill=tk.BOTH, expand=True)
        graph_btn = ttk.Button(btn_frame, text="Show Graph", command=show_graph)
        graph_btn.grid(row=1, column=0, padx=5, pady=5)
        def show_all_graphs():
            w = tk.Toplevel(root); w.title('All Islands Cost Histories')
            fig, axs = plt.subplots(2, 2, figsize=(12,8)); axs = axs.flatten()
            for i, ax in enumerate(axs):
                if i < len(logs):
                    ax.plot(logs[i]); ax.set_title(options[i])
                    ax.set_xlabel('Gen'); ax.set_ylabel('Cost')
                else:
                    ax.axis('off')
            fig.tight_layout()
            c = FigureCanvasTkAgg(fig, master=w); c.draw(); c.get_tk_widget().pack(fill=tk.BOTH, expand=True)
        all_btn = ttk.Button(btn_frame, text="Show All Graphs", command=show_all_graphs)
        all_btn.grid(row=1, column=1, padx=5, pady=5)
    ttk.Button(main_frame, text='Load JSON', command=load_file).grid(row=0, column=0, pady=5, sticky='w')
    ttk.Label(main_frame, text='Population size:').grid(row=1, column=0, sticky='e')
    pop_entry = ttk.Entry(main_frame); pop_entry.insert(0,'100'); pop_entry.grid(row=1, column=1, sticky='w')
    ttk.Label(main_frame, text='Generations:').grid(row=1, column=2, sticky='e')
    gen_entry = ttk.Entry(main_frame); gen_entry.insert(0,'100'); gen_entry.grid(row=1, column=3, sticky='w')
    ttk.Label(main_frame, text='Islands rate:').grid(row=2, column=0, sticky='e')
     iso_ratio_entry = ttk.Entry(main_frame); iso_ratio_entry.insert(0,'0.25'); iso_ratio_entry.grid(row=2, column=1,
sticky='w')
    ttk.Label(main_frame, text='Mutation rate:').grid(row=2, column=2, sticky='e')
     mutation_entry = ttk.Entry(main_frame); mutation_entry.insert(0,'0.02'); mutation_entry.grid(row=2, column=3,
sticky='w')
    start_button = ttk.Button(main_frame, text="Run", command=start_algorithm)
    start_button.grid(row=0, column=2)
    root.mainloop()
if __name__ == '__main__':
    run_gui()
```

Приложение Б

Код программы для автоматического решения транспортных задач

```
import time
import pandas as pd
import numpy as np
from Generator import generate_instance_nd
from Transport_Problem import (
    genetic_algorithm_islands,
    northwest_seed, least_cost_seed, HiGHS_seed, random_seed
)
import sys
sys.stdout.reconfigure(encoding="utf-8")
# Параметры экспериментов
EXPERIMENTS = [
    {'dims': [100, 100],    'num_tasks': 10, 'runs': 10},
    {'dims': [3, 3, 3, 3, 3, 3, 3],    'num_tasks': 10, 'runs': 10},
    {'dims': [10, 10, 10, 10],    'num_tasks': 10, 'runs': 10},
    {'dims': [3, 3],       'num_tasks': 10, 'runs': 10},
    {'dims': [5, 5],       'num_tasks': 10, 'runs': 10},
    {'dims': [7, 7],       'num_tasks': 10, 'runs': 10},
    {'dims': [3, 3, 3],    'num_tasks': 10, 'runs': 10},
    {'dims': [5, 5, 5],    'num_tasks': 10, 'runs': 10},
    {'dims': [7, 7, 7],    'num_tasks': 10, 'runs': 10},
    {'dims': [3, 3, 3, 3],    'num_tasks': 10, 'runs': 10},
    {'dims': [5, 5, 5, 5],    'num_tasks': 10, 'runs': 10},
]
POP_SIZE = 100
GENERATIONS = 100
ISLANDS = 4
SEED_RATIO = 0.25
MUTATION_RATE = 0.02
SEED_METHODS = [
    ('Northwest', northwest_seed),
    ('LeastCost', least_cost_seed),
    ('LP', HiGHS_seed),
    ('Random', random_seed)
]
def run_experiment(exp, balance_margins=False):
    dims = exp['dims']
    num_tasks = exp['num_tasks']
    runs = exp['runs']
    records = []
    for task_id in range(1, num_tasks + 1):
        instance = generate_instance_nd(dims, balance_margins=False, normalize_costs_flag=False)
        constraints = [np.array(m) for m in instance['margins']]
        costs = np.array(instance['costs'])
        for name, method in SEED_METHODS:
            start = time.time()
            plan = method(constraints, costs)
            t = time.time() - start
            cost_val = float(np.sum(plan * costs))
            records.append({
                'dims': tuple(dims),
                'task_id': task_id,
                'method': name,
                'run_id': 0,
                'best_cost': cost_val,
                'time_sec': t
```

```
            })
        for run_id in range(1, runs + 1):
            start = time.time()
            _, best_cost, _ = genetic_algorithm_islands(
                constraints, costs,
                pop_size=POP_SIZE,
                generations=GENERATIONS,
                islands=ISLANDS,
                seed_ratio=SEED_RATIO,
                mutation_rate=MUTATION_RATE
            )
            t = time.time() - start
            records.append({
                'dims': tuple(dims),
                'task_id': task_id,
                'method': 'GA',
                'run_id': run_id,
                'best_cost': float(best_cost),
                'time_sec': t
            })
        print(f"Completed dims={dims}, task {task_id}/{num_tasks}")
    return pd.DataFrame(records)

def main(balance_margins=False):
    all_dfs = []
    for exp in EXPERIMENTS:
        print(f"Running experiment for dims={exp['dims']}")
        df = run_experiment(exp, balance_margins)
        all_dfs.append(df)
    full_df = pd.concat(all_dfs, ignore_index=True)
    full_df.to_excel('multi_test_results.xlsx', index=False)
    print("All experiments done. Results saved to multi_test_results.xlsx")

if __name__ == '__main__':
    main(balance_margins=False)
```

Приложение В

Код программы для генерации транспортных задач

```
import json
import numpy as np
import os
import random
import sys
sys.stdout.reconfigure(encoding="utf-8")
def generate_margins_2d(num_suppliers, num_consumers, total=None, balance_margins=True):
    if balance_margins:
        if total is None:
            total = random.randint(50, 200)
        cuts = sorted(random.sample(range(1, total), num_suppliers + num_consumers - 1))
        parts = [cuts[0]] + [cuts[i] - cuts[i-1] for i in range(1, len(cuts))] + [total - cuts[-1]]
        supplies = parts[:num_suppliers]
        demands = parts[num_suppliers:]
    else:
        supplies = [random.randint(0, 100) for _ in range(num_suppliers)]
        demands = [random.randint(0, 100) for _ in range(num_consumers)]
    return supplies, demands
def generate_costs_2d(num_suppliers, num_consumers, low=1, high=100):
    return np.random.randint(low, high+1, size=(num_suppliers, num_consumers)).tolist()
def normalize_costs(costs):
    arr = np.array(costs, dtype=float)
    minc, maxc = arr.min(), arr.max()
    if maxc == minc:
        return arr.tolist(), (minc, maxc)
    norm = (arr - minc) / (maxc - minc)
    return norm.tolist(), (minc, maxc)
def generate_instance_2d(num_suppliers, num_consumers,
                         total=None, balance_margins=True,
                         cost_low=1, cost_high=100,
                         normalize_costs_flag=False,
                         output_path=None):
    supplies, demands = generate_margins_2d(num_suppliers, num_consumers, total, balance_margins)
    costs = generate_costs_2d(num_suppliers, num_consumers, cost_low, cost_high)
    norm_params = None
    if normalize_costs_flag:
        costs, norm_params = normalize_costs(costs)
    instance = {
        "margins": [supplies, demands],
        "costs": costs,
        "balanced_margins": balance_margins,
        "normalized_costs": normalize_costs_flag,
        "normalization_params": norm_params
    }
    if output_path:
        os.makedirs(os.path.dirname(output_path), exist_ok=True)
        with open(output_path, 'w', encoding='utf-8') as f:
            json.dump(instance, f, ensure_ascii=False, indent=2)
    return instance
def generate_margins_nd(dim_sizes, total=None, balance_margins=True):
    N = len(dim_sizes)
    margins = []
    if balance_margins:
        if total is None:
            total = random.randint(50, 200)
        for size in dim_sizes:
```

```
            cuts = sorted(random.sample(range(1, total), size-1))
            parts = [cuts[0]] + [cuts[i]-cuts[i-1] for i in range(1, len(cuts))] + [total-cuts[-1]]
            margins.append(parts)
    else:
        for size in dim_sizes:
            margins.append([random.randint(0, 100) for _ in range(size)])
    return margins
def generate_costs_nd(dim_sizes, low=1, high=100):
    return np.random.randint(low, high+1, size=tuple(dim_sizes)).tolist()
def generate_instance_nd(dim_sizes,
                         total=None, balance_margins=True,
                         cost_low=1, cost_high=100,
                         normalize_costs_flag=False,
                         output_path=None):
    margins = generate_margins_nd(dim_sizes, total, balance_margins)
    costs = generate_costs_nd(dim_sizes, cost_low, cost_high)
    norm_params = None
    if normalize_costs_flag:
        flat = np.array(costs, dtype=float)
        if flat.max() != flat.min():
            flat_norm = (flat - flat.min()) / (flat.max() - flat.min())
        else:
            flat_norm = flat
        costs = flat_norm.tolist()
        norm_params = (float(flat.min()), float(flat.max()))
    instance = {
        "margins": margins,
        "costs": costs,
        "balanced_margins": balance_margins,
        "normalized_costs": normalize_costs_flag,
        "normalization_params": norm_params
    }
    if output_path:
        os.makedirs(os.path.dirname(output_path), exist_ok=True)
        with open(output_path, 'w', encoding='utf-8') as f:
            json.dump(instance, f, ensure_ascii=False, indent=2)
    return instance
if __name__ == "__main__":
    generate_instance_nd([4,4,4,4], total=500, balance_margins=True,
                                normalize_costs_flag=False,
                                output_path="./data/4d_bal.json")
```